AFRL-RI-RS-TR-2017-218

PROBABILISTIC PROGRAMMING FOR ADVANCED MACHINE LEARNING (PPAML) - DISCRIMINATIVE LEARNING FOR GENERATIVE TASKS (DILIGENT)

VENCORE LABS

NOVEMBER 2017

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2017-218 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
EDWARD A. VERENICH
Work Unit Manager

/ S /
JULIE BRICHACEK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
NOV 2017	FINAL TECHNICAL REPORT	OCT 2013 – AUG 2017

4. TITLE AND SUBTITLE

PROBABILISTIC PROGRAMMING FOR ADVANCED MACHINE LEARNING (PPAML) - DISCRIMINATIVE LEARNING FOR GENERATIVE TASKS (DILIGENT)

5a. CONTRACT NUMBER
FA8750-14-C-0008

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
61101E

6. AUTHOR(S)

Rauf Izmailov, Vladimir Vapnik

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
R17Y

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Vencore Labs
150 Mt Airy Road
Basking Ridge NJ 07920-2021

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RISC
525 Brooks Road
Rome NY 13441-4505

DARPA/12O
675 North Randolph Street
Arlington, VA 22203-2114

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2017-218

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This Final Report summarizes the research conducted in the course of DARPA PPAML program, which was focused on enabling the use of discriminative solvers to solve discriminative tasks specified in probabilistic programs. The research produced two complementary methods of constructive discriminative solvers: model-driven and data-driven ones: research frameworks and software implementations) in both generative and discriminative areas:

- Generative models: a novel framework for most accurate computation of key statistical elements of model-driven problems (such as conditional probability, regression, etc.)
- Discriminative models: a novel framework for capturing domain knowledge in the form of features and kernels for standard data-driven problems (solved in LUPI approaches).

These achievements are described in in this report and in 9 papers published in the course of DARPA PPAML program.

15. SUBJECT TERMS

Machine Learning, Regression, Conditional Probability, Ensemble Learning, Support Vector Machines

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**EDWARD A. VERENICH**
U	U	U	UU	51	19b. TELEPHONE NUMBER (*Include area code)*

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

1 SUMMARY

This Final Report summarizes the research conducted in the course of DARPA PPAML program, which was focused on enabling the use of discriminative solvers to solve discriminative tasks specified in probabilistic programs. The research produced two complementary methods of constructive discriminative solvers: model-driven and data-driven ones:

- Generative models: a novel framework for most accurate computation of key statistical elements of model-driven problems (such as conditional probability, regression, etc.)
- Discriminative models: a novel framework for capturing domain knowledge in the form of features and kernels for standard data-driven problems (solved in LUPI approaches).

These achievements are described in in this report and in 9 papers published in the course of DARPA PPAML program.

2 INTRODUCTION

As explained in Introduction, the focus of our project is *to enable the use of discriminative solvers to solve discriminative tasks specified in probabilistic programs.* We refer to the system that we will research and develop as DIscriminative LEarning for GENerative Tasks (DILEGENT). We achieve this by focusing on two complementary methods of constructive discriminative solvers: model-driven and data-driven ones. Conceptually, we illustrate them in Figure 1. For both methods, our goal is to create a decision rule (lower right corner in Figure 1) based on training data (lower left corner in Figure 1). In model-driven approach, the path to that decision rule consists of two conceptual steps (building a probabilistic model and using it to create a decision rule – upper part of Figure 1), whereas in data-driven approach, that path consists of one direct step (lower part of Figure 1). Both approaches have known advantages and disadvantages, as explained next.

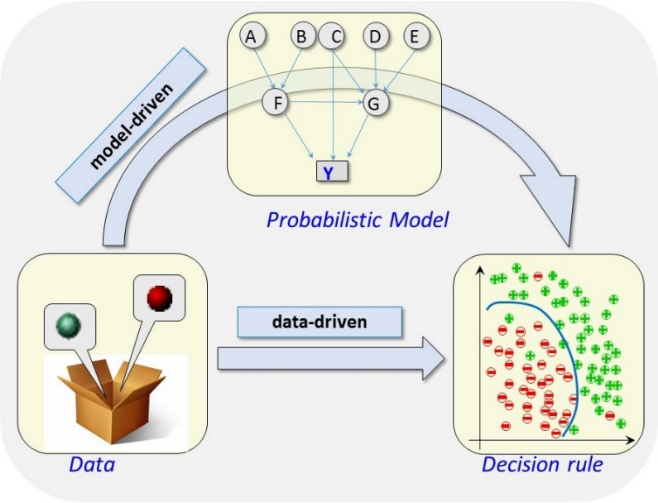

Figure 1. Model-Driven and Data-Driven Approaches.

The model-driven approach is based on the assumption that the general underlying model is specified, and the only tasks that have to be executed sequentially are (1) to estimate parameters of the model, and (2) to construct the decision function based on completely specified model. This is a well-established approach with a variety of mathematical and programming tools available. However, the actual system structure and underlying probabilistic distributions may differ from the simplified model, and the target parameters may be difficult to estimate accurately due to insufficient data, poor diversity of models, ill-posed problems etc. Our goal is to bring some ideas developed in the data-driven approach to improve its performance.

The data-driven approach does not rely on specific models; instead, it is focused on finding the best decision function directly. Since it has one conceptual step instead of two sequential steps used by model-driven approach, it typically deliver better performance (in terms of accuracy / error rate, robustness, etc.). However, by doing that, potentially valuable domain knowledge information, which justifies our other goal of bringing some ideas developed in model-driven approach to improve its performance.

The potential applications of both approaches enhanced in the project include improved performance of a variety of key statistical / machine learning mechanisms, such as Classification, Regression, Ensemble, Recommendation, Ranking, Missing data, Multiple conflicting decisions, Imbalanced data, etc.

The key enabling technology for model-driven approach is a scalable algorithm for solving underlying ill-posed (unstable) integral equations for conditional probability by restricting their solutions to monotonic functions thus converting them into well-posed (stable) ones. The key enabling technology for data-driven approach is a method of encoding knowledge (such as privileged information, structure information, etc.) into additional features before applying standard machine learning algorithms.

As a result of development and testing these technologies in model-driven approach, we implemented conditional probability estimation techniques that produce accurate data-based solutions with improved accuracy by 35% over SoA (standard ensemble methods). Correspondingly, in data-driven approach, we implemented techniques for encoding model-based information into features with improved performance by 40% over SoA (standard SVM and neural networks).

3 METHODS, ASSUMPTIONS AND PROCEDURES

In this section, we present our results on both model-driven and data-driven approaches.

3.1 Model-Driven Approach

Conditional probability is one of central concepts in computational decision making. Indeed, from a decision involves making a choice from a set of possible choices based on input information, for example, in an image classification problem, a label is assigned to a given image by analyzing its pixels. If a decision involves probabilistic classification, it is crucial to be able not only to map the observed data into one of the pre-determined classes, but to do so with varying degrees of confidence.

Current discriminative methods are mostly developed for accurately outputting values (categories, numerical, ordinal values, etc.) as their decisions. This is because they are designed to directly minimize the risk of incorrect values (misclassification, regression value, etc.). They do achieve a good performance at this task because predicting values, say 0 or 1 in classification, turns out to be a simpler task than predicting conditional probability of a particular value. To address the need of predicting conditional probability, these methods employ an ad-hoc post-processing step where the output of the classical discriminative methods is mapped to conditional probabilities. This is both inefficient and erroneous. It is inefficient because it is a multistep process. It is erroneous because the output of the first step, the decision-making step, reduces a multi-dimensional input to a single dimensional output which means there is information loss and there may not be sufficient information to correctly estimate the conditional probability from a single dimensional input.

The ability to accurately and efficiently output conditional probabilities is a natural first step in advancing discriminative methods to answer queries in probabilistic programs by employing them as solvers of probabilistic programs. So, we conducted research on a novel fundamental approach to estimating *conditional probabilities*; within the framework of discriminative learning, these probabilities translate into *probabilistic classification*, i.e., the set of probabilities assigned to possible classifications of any given data point. Our approach aims at providing better estimates of conditional probabilities and is suitable for a wide range of problems in decision theory and machine learning.

In our papers [1] [2] [3], we focused on main targets of statistical inference theory is estimation (from the data) of specific models of random events, namely:

1. conditional probability function;
2. conditional density function;
3. regression function;
4. density ratio function.

These models can be represented in the following manner. Let $F(x)$ be a cumulative distribution function of random variable x. We call non-negative function $p(x)$ the probability density function if

$$\int_{-\infty}^{x} p\,(x^*)dx^* = F(x) \tag{1}$$

Similarly, let $F(x,y)$ be the joint probability distribution function of variables x and y. We call non-negative $p(x,y)$ the joint probability density function of two variables x and y if

$$\int_{-\infty}^{y}\int_{-\infty}^{x} p\,(x^*,y^*)dx^*dy^* = F(x,y). \tag{2}$$

Let $p(x,y)$ and $p(x)$ be probability density functions for pairs (x,y) and vectors x. Suppose that $p(x) > 0$. The function

$$p(y|x) = \frac{p(x,y)}{p(x)} \tag{3}$$

is called the *Conditional Density Function*. It defines, for any fixed $x = x_0$, the probability density function $p(y|x = x_0)$ of random value $y \in R^1$. The estimation of the conditional density function from data

$$(y_1, X_1), \ldots, (y_\ell, X_\ell) \tag{4}$$

is the most difficult problem in our list of statistical inference problems.

Along with estimation of the conditional density function, the important problem is to estimate the so-called *Conditional Probability Function*. Let variable y be discrete, say, $y \in \{0,1\}$. The function defined by the ratio

$$p(y=1|x) = \frac{p(x,y=1)}{p(x)}, \quad p(x) > 0 \tag{5}$$

is called *Conditional Probability Function*. For any given vector $x = x_0$, this function defines the probability that y is equal to one; correspondingly, $p(y=0|x=x_0) = 1 - p(y=1|x=x_0)$. The problem is to estimate the conditional probability function, given data (5) where $y \in \{0,1\}$.

Estimation of the conditional density function is a difficult problem; a much easier problem is the problem of estimating the so-called *Regression Function* (conditional expectation of the variable y):

$$r(x) = \int y\, p(y|x) dy, \tag{6}$$

which defines expected value $y \in R^1$ for a given vector x.
We also consider a problem, which is important for applications: estimating the ratio of two probability densities. Let $p_{\text{num}}(x)$ and $p_{\text{den}}(x) > 0$ be two different density functions (subscripts *num* and *den* correspond to numerator and denominator of the density ratio). Our goal is to estimate the function

$$R(x) = \frac{p_{num}(x)}{p_{den}(x)} \tag{7}$$

given iid data

$$X_1, \ldots, X_{\ell_{\text{den}}}, \tag{8}$$

distributed according to $p_{\text{den}}(x)$, and iid data

$$X'_1, \ldots, X'_{\ell_{\text{num}}}, \tag{9}$$

distributed according to $p_{\text{num}}(x)$.

Next, we describe our direct settings for these four statistical inference problems.

By definition, conditional density $p(y|x)$ is the ratio of two densities

$$p(y|x) = \frac{p(x,y)}{p(x)}, \quad p(x) > 0 \tag{10}$$

or, equivalently,

$$p(y|x)p(x) = p(x,y). \tag{11}$$

This expression leads to the following equivalent one:

$$\int\int \theta\,(y-y')\theta(x-x') f(x',y') dF(x') dy' = F(x,y) \tag{12}$$

where $f(x,y) = p(y|x)$, function $F(x)$ is the cumulative distribution function of x and $F(x,y)$ is the joint cumulative distribution function of x and y.
Therefore, our setting of the condition density estimation problem is as follows:

- *Find the solution of the above integral equation in the set of nonnegative functions $f(x,y) = p(y|x)$ when the cumulative probability distribution functions $F(x,y)$ and $F(x)$ are unknown but iid data*

$$(y_1, X_1), \ldots, (y_\ell, X_\ell) \tag{13}$$

are given.

In order to solve this problem, we use empirical estimates

$$F_\ell(x,y) = \frac{1}{\ell}\sum_{i=1}^{\ell} \theta\,(y-y_i)\theta(x-X_i), \tag{14}$$

$$F_\ell(x) = \frac{1}{\ell}\sum_{i=1}^{\ell} \theta\,(x - X_i) \tag{15}$$

of the unknown cumulative distribution functions $F(x, y)$ and $F(x)$. Therefore, we have to solve an integral equation where not only its right-hand side is defined approximately (we can only deal with $F_\ell(x, y)$ instead of $F(x, y)$), but also the data-based approximation

$$A_\ell f(x, y) = \int\int \theta\,(y - y')\theta(x - x')f(x', y')dy'dF_\ell(x') \tag{16}$$

is used instead of the exact integral operator

$$Af(x, y) = \int\int \theta\,(y - y')\theta(x - x')f(x', y')dy'dF(u'). \tag{17}$$

Taking into account empirical estimates $F_\ell(x, y)$ and $F_\ell(x)$, our goal is thus to find the solution of approximately defined equation

$$\sum_{i=1}^{\ell} \theta\,(x - X_i)\int_{-\infty}^{y} f\,(X_i, y')dy' \approx \frac{1}{\ell}\sum_{i=1}^{\ell} \theta\,(y - y_i)\theta(x - X_i). \tag{18}$$

According to the definition of conditional probability,

$$\int_{-\infty}^{\infty} p\,(y|x)dy = 1, \quad \forall x \in X. \tag{19}$$

Therefore, the solution of our equation has to satisfy the constraint $f(x, y) \geq 0$ and the constraint

$$\int_{-\infty}^{\infty} f\,(y', x)dy' = 1, \quad \forall x \in X. \tag{20}$$

We call this setting *the direct constructive setting* since it is based on direct definition of conditional density function above and uses theoretically justified approximations $F_\ell(x, y)$ and $F_\ell(x)$ of the corresponding unknown functions. In other words, direct constructive approach that we developed consists of replacing the unknown cumulative distribution functions we use their empirical approximations

$$F_\ell(x) = \tfrac{1}{\ell}\textstyle\sum_{i=1}^{\ell} \theta\,(x - X_i), \tag{21}$$

$$F_\ell(x, y = 1) = p_\ell F_\ell(x|y = 1) = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i\,\theta(x - X_i), \tag{22}$$

where p_ℓ is the ratio of the number of examples with $y = 1$ to the total number ℓ of the observations. These empirical approximations are then used as a replacement of original functions in the corresponding integral equations and solving the resulting systems by regularization approach and minimization of discrepancy between right-hand side and left-hand sides

Therefore, one has to solve our original integral equation with approximately defined right-hand side and approximately defined operator

$$A_\ell f(x) = \tfrac{1}{\ell}\textstyle\sum_{i=1}^{\ell} \theta\,(x - X_i)f(X_i). \tag{23}$$

Since the probability takes values between 0 and 1, our solution has to satisfy the bounds

$$0 \leq f(x) \leq 1, \quad \forall x \in X. \tag{24}$$

Also,

$$\int f\,(x)dF(x) = p(y = 1), \tag{25}$$

where $p(y = 1)$ is the probability of $y = 1$.

By definition, regression is the conditional mathematical expectation

$$r(x) = \int y\, p(y|x) dy = \int y \frac{p(x,y)}{p(x)} dy. \tag{26}$$

This can be rewritten in the form

$$r(x)p(x) = \int y\, p(x,y) dy. \tag{27}$$

Thus we obtain the equivalent equation

$$\int \theta\,(x - x')r(x')dF(x') = \int \theta\,(x - x') \int y\, dF(x', y'). \tag{28}$$

Therefore, the direct constructive setting of regression estimation problem is as follows:

In a given set of functions $r(x)$, find the solution of integral equation (6) if cumulative probability distribution functions $F(x, y)$ and $F(x)$ are unknown but iid data $(y_1, X_1), \ldots, (y_\ell, X_\ell)$ are given.

As before, instead of these functions, we use their empirical estimates. That is, we construct the approximation

$$A_\ell r(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} \theta\,(x - X_i) r(X_i) \tag{29}$$

instead of the actual operator, and the approximation of the right-hand side

$$F_\ell(x) = \frac{1}{\ell} \sum_{j=1}^{\ell} y_j\, \theta\big(x - X_j\big) \tag{30}$$

instead of the actual right-hand side in the above integral equation, based on the observation data

$$(y_1, X_1), \ldots, (y_\ell, X_\ell), \quad y \in R^1,\ x \in X. \tag{31}$$

Let $F_{\text{num}}(x)$ and $F_{\text{den}}(x)$ be two different cumulative distribution functions defined on $X \subset R^d$ and let $p_{\text{num}}(x)$ and $p_{\text{den}}(x)$ be the corresponding density functions. Suppose that $p_{\text{den}}(x) > 0, x \in X$. Consider the ratio of two densities:

$$R(x) = \frac{p_{num}(x)}{p_{den}(x)}. \tag{32}$$

The problem is to estimate the ratio $R(x)$ when densities are unknown, but iid data

$$X_1, \ldots, X_{\ell_{\text{den}}} \sim F_{\text{den}}(x), \tag{33}$$

generated according to $F_{\text{den}}(x)$, and iid data

$$X'_1, \ldots, X'_{\ell_{\text{num}}} \sim F_{\text{num}}(x), \tag{34}$$

generated according to $F_{\text{num}}(x)$, are given.
As before, we introduce the constructive setting of this problem: solve the integral equation

$$\int \theta\,(x - u) R(u) dF_{\text{den}}(u) = F_{\text{num}}(x) \tag{35}$$

when cumulative distribution functions $F_{\text{den}}(x)$ and $F_{\text{num}}(x)$ are unknown, but the data drawn from these distributions are given. As before, we approximate the unknown cumulative distribution functions $F_{\text{num}}(x)$ and $F_{\text{den}}(x)$ using empirical distribution functions

$$F_{\ell_{\text{num}}}(x) = \frac{1}{\ell_{num}} \sum_{j=1}^{\ell_{\text{num}}} \theta\left(x - X'_j\right) \tag{36}$$

for $F_{\text{num}}(x)$, and

$$F_{\ell_{\text{den}}}(x) = \frac{1}{\ell_{den}} \sum_{j=1}^{\ell_{\text{den}}} \theta\left(x - X_j\right) \tag{37}$$

for $F_{\text{den}}(x)$.
Since $R(x) \geq 0$ and $\lim_{x\to\infty} F_{\text{num}}(x) = 1$, our solution has to satisfy the constraints

$$R(x) \geq 0, \quad \forall x \in X, \tag{38}$$

$$\int R(x) dF_{\text{den}}(x) = 1. \tag{39}$$

Therefore, all main empirical inference problems described above (conditional probability, regression, density ratio) can be represented via (multidimensional) Fredholm integral equation of the first kind with approximately defined elements. Although approximations converge to the true functions, these problems are computationally difficult due to their ill-posed nature. Thus they require rigorous solutions. Various statistical methods exist for solving these inference problems. Our goal is to find general rigorous solutions that take into account all the available characteristics of the problems.

We now present a general form for all statistical inference problems.

Consider the multidimensional Fredholm integral equation

$$\int \theta(z - z') f(z') dF_A(z') = F_B(z), \tag{40}$$

where the kernel of operator equation is defined by the step function $\theta(z - z')$, the cumulative distribution functions $F_A(z)$ and $F_B(z)$ are unknown but the corresponding iid data

$$Z_1, \ldots, Z_{\ell_A} \sim F_A(z) \tag{41}$$

$$Z_1, \ldots, Z_{\ell_B} \sim F_B(z) \tag{42}$$

are given. In the different inference problems, the elements $f(z), F_A(z), F_B(z)$ of the equation have different meanings:

- In the problem of **conditional density** estimation, vector z is the pair (x, y), the solution $f(z)$ is $p(y|x)$, the cumulative distribution function $F_A(z)$ is $F(x)$ and the cumulative distribution function $F_B(z)$ is $F(x, y)$.
- In the problem of **conditional probability** $p(y = 1|x)$ estimation, vector z is x, the solution $f(z)$ is $p(y = 1|x)$, the cumulative distribution function $F_A(z)$ is $F(x)$, the cumulative distribution function $F_B(z)$ is $F(x|y = 1)p(y = 1)$, where $p(y = 1)$ is the probability of class $y = 1$.
- In the problem of **density ratio** estimation, the vector z is x, the solution $f(z)$ is $p_{num}(x)/p_{den}(x)$, the cumulative function $F_A(z)$ is $F_{num}(x)$, the cumulative function $F_B(z)$ is $F_{den}(x)$.
- In the problem of **regression** $R(x) = \int y\, p(y|x) dy$ estimation, the vector z is (x, y), where $y \geq 0$, the solution $f(z)$ is $\hat{y}^{-1} R(x)$, $(R(x) = \int y\, p(y|x) dy)$, the cumulative function $F_A(z)$ is $F(x)$, the cumulative function $F_B(z)$ is $\hat{y}^{-1} \int \theta(x' - x') y' dF(x', y')$.

Since statistical inference problems have the same kernel of the integral equations (i.e., the step-function) and the same right-hand side (i.e., the cumulative distribution function), it allows us to develop a common standard method for solving all inference problems.

As a result, we have developed [1] [2] [3] the fundamental approach for generative models that allows for most accurate estimation of the key statistical quantities. The approach has been tested on a number of synthetic examples, consistently delivering performance that exceeds that of standard methods.

Figure 2 illustrates both classical method and DILEGENT method applied to the same problem of estimating the conditional probability (true conditional probability is shown as blue line, and its estimate – as black line) based on one-dimensional samples of two classes (shown as red and green markers around horizontal axis), consisting of 48, 96, 192 and 384 elements. Our novel approach is shown in the right column in Figure 2, while the classical approach is shown in the left column.

As Figure 2 illustrates, with the increase of training sample size (from 48 to 384) the resulting approximations converge to the true conditional probability converge, but our approach does it faster and more accurately than the classical approach.

In the context of SVM, the conditional probability of SVM outputs can be further analyzed in the following manner. As Platt [4] observed, the smaller is the (negative) score s_i for vector z_i, the closer is the conditional probability $P(y=1|s_i)$ to zero and, the larger is the (positive) score s_i, the closer is the conditional probability $P(y=1|s_i)$ to one. Platt introduced a method for mapping SVM scores into values of conditional probability based on two hypotheses, a general one and a special one.

The general hypothesis: Conditional probability function $p(y=1|s)$ is a monotonic function of variable s.

The special hypothesis: Conditional probability function can be approximated well with sigmoid functions with two parameters:

$$P(y=1|s)=\frac{1}{1+\exp\{-As+B\}},\quad A,B\in R^1. \tag{43}$$

Using the maximum likelihood technique, [4] introduced effective methods to estimate both parameters A,B (see [5]).

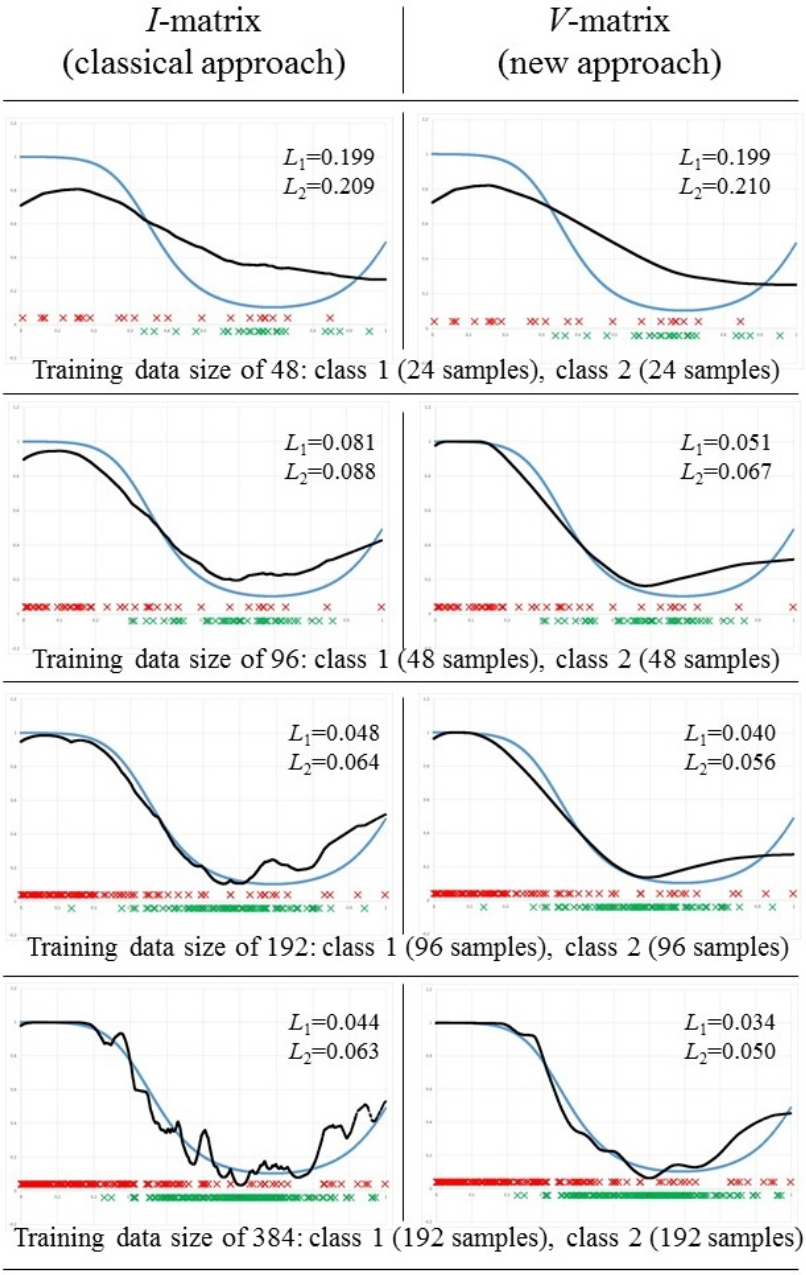

Figure 2. Model-Driven and Data-Driven Approximations of Conditional Probability.

Platt's approach was shown to be useful for calibration of SVM scores. Nevertheless, this method has certain drawbacks: even if the conditional probability function for SVM is monotonically increasing, it does not necessarily have the form of a two-parametric sigmoid function. It is easy to construct examples where suggested sigmoid function does not approximate well the desired monotonic conditional probability function.

The one-dimensional problem mentioned in the previous paragraph has the following form: given pairs (values s_i of SVM scores and corresponding classifications y_i)

$$(s_1,y_1),\ldots,(s_\ell,y_\ell), \tag{44}$$

find an accurate approximation of the monotonic conditional probability function $p(y=1|s)$. Further, we

describe a technique for construction of a monotonic approximation of the desired function. This approximation provides a more accurate estimate than the one based on sigmoid functions. We also consider a more general (and more important) problem than this one-dimensional one. Suppose we have d different SVMs, solving the same classification problem. Also, suppose that the probability of class y=1 given scores $s=(s^1,\ldots,s^d)$ of d SVMs is a multidimensional monotonic conditional probability function: for any coordinate k and any fixed values of the other coordinates $(s^1,\ldots,s^{k-1},s^{k+1},\ldots s^d)$, the higher is the value of score s^k, the higher is the probability $P(y=1|s)$.

The goal is to find a method for estimation of the *monotonic* conditional probability function $P(y=1|s)$ for multidimensional vectors $s=(s^1,\ldots,s^d)$; that is, to combine, in a single probability value, the results of multiple (namely, d) SVMs. We show that estimating conditional probability function in a set of monotonic functions has a significant advantage over estimating conditional probability function in a general, non-monotonic set of functions: it forms a *well-posed* problem rather than an *ill-posed* problem.

The decision rule for a two-class pattern recognition problem can be obtained using the estimated conditional probability function $P(y=1|s)$ as

$$y=\Theta\left(P(y=1|s)-\frac{1}{2}\right). \tag{45}$$

It is important to note that, in classical machine learning literature, there are *ensemble methods* that combine several rules (see [6], [7], [8]). The difference between ensemble rules and synergy rules is in the following:

1) Ensemble rule is a result of structural combination (such as voting or weighted aggregation) of several classification rules.

2) Synergy rule defines the *optimal* solution to the problem of combining several scores of *monotonic rules*. It is based on effective methods of conditional probability estimation in the set of monotonic functions.

3) Synergy rule is constructed only for *monotonic rules* (such as SVM) in contrast to ensemble rule which combines *any* rules. Synergy is the property of monotonicity of the solution.

Our goal is to minimize conditional probability in the set of monotonically increasing functions. We do this by using expansion of desired function on kernels that generate splines with infinite number of knots (INK-spline) of degree zero. The reason we use these kernels is that they enable an efficient and straightforward construction of multidimensional monotonic functions; it is possible that some other kernels might be used for that purpose as well.

According to the definition in the one-dimensional case, splines of degree r with m knots are defined by the expansion (here, we assume that $0\le x\le 1$)

$$S(x|r,m)=\sum_{s=0}^{r} c_s x^s+\sum_{k=0}^{m} e_k (x-a_k)_+^r, \tag{46}$$

where

$$(x-a_k)_+^r=\begin{cases}(x-a_k)^r \text{ if } x-a_k\ge 0\\ 0 \quad \text{otherwise}\end{cases} \tag{47}$$

We generalize this representation using infinite number of knots:

$$S_{\infty}(x)=\sum_{s=0}^{r} c_s x^s+\int_0^{\infty} g(\tau)(x-\tau)_+^r d\tau. \tag{48}$$

Following the approach from [9], [10], we define the kernel with infinite number of knots (INK-spline) of degree r for expansion of the function of one variable $x\geq 0$ in the form

$$K_r(x_i,x_j)=\int_0^{\infty}(x_i-\tau)_+^r(x_j-\tau)_+^r d\tau=\sum_{k=0}^{r}\frac{C_r^k}{2r-k+1}[\min\{x_i,x_j\}]^{2d-k+1}|x_i-x_j|^k \tag{49}$$

(here we modified the definition of INK-kernel from [10] by omitting its polynomial portion).

For $r=0$, the INK-spline kernel has the form

$$K_0(x_i,x_j)=\min\{x_i,x_j\}; \tag{50}$$

for $r=1$, the INK-spline kernel has the form

$$K_1(x_i,x_j)=\frac{1}{3}\left(\min\{x_i,x_j\}\right)^3+\frac{1}{2}\left(\min\{x_i,x_j\}\right)^2|x_i-x_j|. \tag{51}$$

In the multidimensional case, the INK-spline of degree r is defined as

$$K_r(x_i,x_j)=\prod_{k=1}^{d}K_r(x_i^k,x_j^k),\quad x=(x^1,\ldots,x^d). \tag{52}$$

In order to find a monotonic solution, we use our method for estimating conditional probability function with INK-spline kernel of degree zero with additional ℓ monotonicity constraints. That is, we have to minimize the functional

$$W=(K\Lambda+b\mathbf{1}_{\ell})^T V(K\Lambda+b\mathbf{1}_{\ell})-2(K\Lambda+b\mathbf{1}_{\ell})^T VY+\gamma_{\ell}\Lambda^T K\Lambda \tag{53}$$

(here coordinates of vector Y are $y_i\in\{-1,+1\}$ subject to $\ell+1$ inequality constraints

$$\Lambda^T\tau(0)\geq 0,\Lambda^T\tau(x_j)\geq 0, j=1,\ldots,l \tag{54}$$

Let $x\geq 0$. Then, in order to construct the conditional probability in the set of non-negative monotonic functions bounded by the value 1, we have to enforce the constraint $P(y=1|x)\leq 1$. Thus, taking into account nonnegativity and monotonicity constraints, we add the constraint

$$\Lambda^T K(x=1)+b=\Lambda^T\vec{\mathbf{x}}+b\leq 1, \tag{55}$$

where $\vec{\mathbf{x}}=(x_1,\ldots x_{\ell})^T$. Using L_2-norm SVM for estimating monotonic conditional probability function, we minimize the functional

$$W(\Lambda)=\Lambda^T KK\Lambda-2\Lambda^T KY+\gamma_{\ell}\Lambda^T K\Lambda, \tag{56}$$

with coordinates of Y are in [0,1] subject to $\ell+2$ inequality constraints described above. In multidimensional case, where we can assume (by proper normalization) that $H\in[0,1]^d$. We consider the solution of the equation in the form

$$f(x)=\sum_{i=1}^{\ell}\alpha_i K(x_i,x)+b, \tag{57}$$

where the kernel generating d-dimensional INK-spline of degree zero has the multiplicative form

$$K(x_i,x)=\prod_{k=1}^{d}\min(x_i^k,x^k). \tag{58}$$

Along with functions defined by (multiplicative) INK-spline kernels of degree zero that can construct approximations to monotonic functions, we consider functions defined by the additive kernel (which is a sum of one-dimensional kernels)

$$f(x)=\sum_{i=1}^{\ell}\sum_{k=1}^{d}\alpha_i^k\min(x_i^k,x^k)+b. \tag{59}$$

In order to find $d\times\ell$ coefficients α_i^k of expansion in estimating conditional probability function in the direct setting, we minimize the functional

$$R(\Lambda_1,\ldots\Lambda_d,b)=\left[\sum_{k=1}^{d}K_k\Lambda_k+b\mathbf{1}_\ell\right]^T V\left[\sum_{k=1}^{d}K_k\Lambda_k+b\mathbf{1}_\ell\right]- \tag{60}$$

$$2\left[\sum_{k=1}^{d}K_k\Lambda_k+b\mathbf{1}_\ell\right]^T VY+\gamma\sum_{k=1}^{d}(\Lambda_k^T K_k\Lambda_k)$$

subject to $d\times(\ell+1)$ inequality constraints

$$\begin{aligned}\frac{\partial f(x_j;\alpha)}{\partial x^k}&=\sum_{i=1}^{\ell}\alpha_i^k\Theta(x_i^k-x_j^k)=\Lambda_k^T\tau(x_j^k)\geq 0;\quad j=1;\ldots;\ell;\ k=1;\ldots;d;\\ \frac{\partial f(\vec{0};\alpha)}{\partial x^k}&=\sum_{i=1}^{\ell}\alpha_i^k=\Lambda_k^T\tau(\vec{0}^k)\geq 0;\quad k=1;\ldots;d;\end{aligned} \tag{61}$$

where we have denoted by Λ_k the ℓ-dimensional vector of $\alpha_i^k=(\alpha_1^k,\ldots,\alpha_l^k), k=1,\ldots,d$, by K_k the $(\ell\times\ell)$-dimensional matrix of elements $K_k(x_i^k,x_j^k)=\min(x_i^k,x_j^k)$, and by $\tau(x_j^k)=(\Theta(x_1^k-x_j^k),\ldots,\Theta(x_l^k-x_j^k))^T, j=1,\ldots,l, k=1,\ldots,d$ we have denoted the $d\times\ell$ vectors of dimensionality ℓ.

Let vector $x=(x^1,\ldots,x^d)$ have bounded coordinates

$$0\leq x^k\leq c_k,\ k=1,\ldots,d \quad . \tag{62}$$

Since conditional probability does not exceed 1, we need one more constraint $P(y=1|c_1,\ldots,c_d)\leq 1$. That is, we have to add the constraint

$$\sum_{k=1}^{d}\Lambda_k^T\mathbf{X}^k+b\leq 1, \tag{63}$$

where we have denoted $\mathbf{X}^k=(x_1^k,\ldots,x_\ell^k)^T$. A function satisfying the described conditions is monotonic.

In order to estimate a multidimensional monotonic function using multiplicative kernel, one has to solve a quadratic optimization problem of order ℓ subject to $N=\ell d$ inequality constraints.

With additive kernel, one has to estimate $d\times(\ell+1)$ parameters under $d\times(\ell+1)$ constraints. To decrease the computation amount:

1. One can replace *V*-matrix with *I*-matrix.
2. For additive kernel, one can estimate multidimensional conditional probability function in the *restricted set of functions* where $\alpha_i^t=\alpha_i$, for some or for all *t*.
3. One can consider linear structure of the solution using *d* one-dimensional estimates of conditional probability $P(y=1|s^t)$ obtained by solving one-dimensional estimation problems and then approximate the multidimensional conditional probability function as

$$P(y=1|s^1,\ldots,s^d)=\sum_{t=1}^{d}\beta_t P(y=1|s^t), \tag{64}$$

where its weights $\beta_t\geq 0,\ \sum\beta_i=1$ are computed by solving an *d*-dimensional quadratic optimization problem under *d*+1 constraints. That optimization problem is formulated as follows: minimize the functional

$$B^TPVPB-2B^TPVY+\gamma B^TB \tag{65}$$

subject to the constraints

$$B\geq 0,\ B^T\mathbf{1}_\ell=1, \tag{66}$$

where we have denoted by *B* vector of coefficients $B=(\beta_1,\ldots,\beta_d)^T$, by *P* the $(d\times\ell)$-dimensional matrix $P=p(x_i^t), t=1,\ldots,d; i=1,\ldots,l.$

.

We now construct several examples of synergy rules for SVMs where we use the same training set both for constructing SVM rules $s_k=s_t(x), t=1,..,d$ for estimating the conditional probability $P(y=1|s_1,\ldots,s_d)$.

Suppose that our rules were constructed using different SVM kernels $K_t(x,y)$ and the same training set

$$(x_1,y_1),\ldots,(x_l,y_l) \tag{67}$$

and let $s_1^t,\ldots,l_j^t, t=1,\ldots,d$ be the scores $s^t=f_t(x)$ obtained using vectors *x*.

Note that these scores are statistically different from the scores obtained using ℓ elements of test set (support vectors s^* are biased: in the separable case, all $|s^*|=1$). Therefore, it is reasonable to use scores obtained in the procedure of *k*-fold cross-validation for estimating parameters of SVM algorithm.

Also, note that while individual components of the same *d*-dimensional vector $S^t=(s_1^t,\ldots,s_d^t)$ are interdependent, the vectors S^t themselves are not (they are i.i.d), so the general theory developed in the

previous sections is applicable here for computing conditional probabilities.

We now consider several examples of synergy of *d* SVM rules obtained under different circumstances:

1. Synergy of *d* rules obtained using the same training data but different kernels.
2. Synergy of *d* rulse obtained using different training data but the same kernel.
3. Synergy of *d* classes classification problem using *d one versus the rest* of the rules.

First, we show that the accuracy of classification using synergy of SVM rules that use different kernels can be much higher than the accuracy of a rule based on any kernel. The idea of using several SVMs as ensemble SVM (such as [11]) was used in the past for providing improved classification performance; however, these approaches did not leverage the main monotonicity property of SVM. The effect of synergy, which is estimated by the number of additional training examples in training data required to achieve comparable to synergy level of accuracy, can be significant.

We selected the following 9 calibration data sets from UCI Machine Learning Repository [12]: Covertype, Adult, Tic-tac-toe, Diabetes, Australian, Spambase, MONK's-1, MONK's-2, and Bank marketing. Our selection of these specific data sets was driven by the desire to ensure statistical reliability of targeted estimates, which translated into availability of relatively large test data set (containing at least 150 samples). Specific breakdowns for the corresponding training and test sets are listed in Table 1. For each of these 9 data sets, we constructed 10 random realizations of training and test data sets; for each of these 10 realizations, we trained three SVMs with different kernels: with RBF kernel, with INK-Spline kernel, and with linear kernel. The averaged test errors of the constructed SVMs are listed in Table 2.

Table 1. Calibration Data Sets from UCI Machine Learning Repository.

Data set	Training	Test	Features
Covertype	300	3000	54
Adult	300	26147	123
Tic-tac-toe	300	658	27
Diabetes	576	192	8
Australian	517	173	14
Spambase	300	4301	57
MONK's-1	124	432	6
MONK's-2	169	432	6
Bank	300	4221	16

Constructed SVMs provide binary classifications *y* and scores *s*. Additional performance improvements are possible by intelligent leveraging of the results of these classifications.

We compared our approach with the baseline method of voting on classification results of all three classifications obtained from three different kernels (since we had odd number of kernels, we did not need any tie-breaking in that vote). The first column of Table 2 shows the averaged test errors of that voting approach.

The second column of Table 2 shows the averaged test errors of our synergy approach. Specifically, the data in the second column are based on constructing a 3-dimensional monotonic conditional probability

function from RKHS associated with additive kernel, on triples of SVM scores *s*. In this column, we assigned the classification labels *y* based on the sign of the difference between 3-dimensional conditional probability and the threshold value 1/2.

The last column of Table 2 contains relative performance gain (i.e., relative decrease of error rate) delivered by the proposed synergy approach over the benchmark voting algorithm.

Table 2. Synergy of SVMs with RBF, INK-Spline, and Linear Kernels.

Data set	Voting	Synergy	Gain
Covertype	27.83%	28.96%	-4.05%
Adult	20.07%	19.08%	4.93%
Tic-tac-toe	1.95%	1.75%	10.16%
Diabetes	24.53%	23.39%	4.67%
Australian	12.02%	12.54%	-4.33%
Spambase	8.96%	8.44%	5.80%
MONK's-1	22.80%	20.16%	11.57%
MONK's-2	19.31%	16.23%	15.95%
Bank	12.79%	11.73%	8.29%

The results demonstrate the consistent performance advantage of synergy approach over its empirical alternative in most of the cases (for 7 data sets out of 9); for some data sets this advantage is relatively small, but for others it is substantial (in relative terms).

This substantial performance improvement of synergy can be also viewed as a viable alternative to brute force approaches relying on accumulation of (big) data. Indeed, for the already considered Adult data set, we compared results of our synergy approach on a training data set consisting of 300 samples to an alternative approach relying on training SVM algorithms on larger training data sets. Specifically, we trained SVMs with RBF kernel and INK-Spline kernel on Adult data sets containing 1,000 and 3,000 samples. The results, shown in Table 3, suggest that synergy of two rules, even on training data set of limited size, can be better that straightforward SVMs on training data sets of much larger sizes (in this example, equivalent to the increase of training sample by more than a factor of 10).

Table 3. Synergy Versus Training Size Increase: Ensemble.

Training size	300	1000	3000
RBF	20.95%	19.21%	18.49%
INK-Spline	19.77%	18.72%	18.38%
Synergy	17.92%	-	-

Suppose now we are dealing with "big data" situation, where the number L of elements in the training data set

$$(x_1,y_1),...,(x_L,y_L), \tag{68}$$

is large. Consider the SVM method that uses a universal kernel. A universal kernel (for example, RBF) can approximate well any bounded continuous function. Generally speaking, with the increase of size ℓ of training data, the expected error rate of the obtained SVM rule monotonically converges to the Bayesian rule (here the expectation is taken both over the rules obtained from different training data of the same size ℓ and over test

data). The typical *learning curve* shows the dependence of that expected error rate on the size ℓ of training data as a hyperbola-looking curve consisting of two parts: the beginning of the curve, where the error rate falls steeply with the increase of ℓ, and the tail of the curve, where the error rate slowly converges to the Bayesian solution. Suppose that the transition from the "steeply falling" part of the curve to the "slowly decreasing" part of the curve (sometimes referred to as the "knee" of the curve) occurs for some ℓ^*. Assuming that large number L in is greater than ℓ^*, we partition the training data into J subsets containing ℓ elements each (here $L=J\ell$ and $\ell>\ell^*$ as well). On each of these J disjoint training subsets we construct its own SVM rule (independent of other rules). For each of these SVM rules, we construct its own one-dimensional monotonic conditional probability function $P_t(y=1|s^t)$, $t=1,\ldots,J$.

Then, using these J one-dimensional monotonic condition probability functions, we construct the J-dimensional ($s=(s^1,...,s^J)$) conditional probability function as follows:

$$P_{syn}(y=1|s)=\frac{1}{J}\sum_{t=1}^{J}P_t(y=1|s^t). \tag{69}$$

The Synergy decision rule in this case has the form

$$y=\Theta\left(P_{syn}(y=1|s)-\frac{1}{2}\right). \tag{70}$$

Note that above conditional probability function forms an unbiased estimate of the values of learning curve describing conditional probability for training data of (different) size ℓ. Since the training data for different t are independent, the averaging of J conditional probability values decreases the variance of resulting conditional probability by a factor of J. In this approach, by choosing an appropriate value of ℓ, one can optimally solve the bias-variance dilemma.

To illustrate this approach, we again used Adult data set. Specifically, we trained SVMs with RBF kernel on Adult data sets containing 900, 1,000 and 3,000 samples. For the first of these samples (containing 900 elements), we also executed the following procedure: we split it into three subsets containing 300 elements each, trained RBF SVM on each of them, and then constructed two combined decision rules: (1) voting on the labels of three auxiliary SVMs, and (2) synergy of three SVMs as described in this section. The results, shown in Table 4, suggest that Synergy of rules on disjoint data sets can be better that straightforward SVMs on training data sets of much larger sizes (in this example, equivalent to the increase of training sample by a factor of 3).

Table 4. Synergy Versus Training Size Increase: Bagging.

Training size	300	300	300	900	1000	3000
RBF SVM	20.77%	19.06%	21.40%	20.01%	19.21%	18.49%
Voting on 3 subsets	N/A	N/A	N/A	19.44%	-	-
Synergy on 3 subsets	N/A	N/A	N/A	18.52%	-	-

Comparison of Table 3 and Table 4 suggests that synergy of SVMs with different SVM kernels obtained on the same data set may be more beneficial (equivalent to ten-fold increase of training sample size) than the synergy of SVMs with the same kernel obtained on different subsets of that data set (equivalent to three-fold increase of training sample size).

Thus it is reasonable to assume that, for big data set, Synergy of SVM rules obtained on different training data and Synergy of SVM rules with different kernels can be unified to create an even more accurate synergy rule. This unification can be implemented in the following manner.

Consider d kernels $K_r(x,x')$, $k=1,\ldots,d$. For each of these kernels, we construct the corresponding condition probability function

$$P_{syn}(y=1|s(r))=\frac{1}{J}\sum_{t=1}^{J}P_t(y=1|s^t(r)), \tag{71}$$

where we have denoted by $P_t(y=1|s^t(r))$ the conditional probability function estimated for the rule with kernel $K_r(x,x')$ and for the jth subset of training data with the fixed t. Let introduce the vector $p=(p^1,...,p^r)$ where

$$p^r=P_{syn}(y=1|s(r)), \quad r=1,\ldots,d. \tag{72}$$

Using these vectors, we estimate the corresponding d-dimensional conditional probability function

$$P_{syn}(y=1|p)=P_{syn}(y=1|p^1,...,p^d) \tag{73}$$

The resulting double reinforced Synergy rule has the form

$$y=\Theta\left(P_{syn}(y=1|p)-\frac{1}{2}\right). \tag{74}$$

We now consider for multi-class classification – an important problem in pattern recognition. In contrast to methods for constructing two-class classification rules, which have solid statistical justifications, existing methods for constructing $d>2$ class classification rules are based on heuristics.

One of the most popular heuristics, *one versus rest* (OVR), suggests first to solve the following d two-class classification problems: in problem number k (where $k=1,\ldots,d$), the examples of class k are considered as examples of the first class and examples of the all other classes $1,\ldots,(k-1),(k+1),\ldots,d$ are considered as the second class. Using OVR approach, one constructs d different two-class classification rules

$$y=\Theta(f_k(x)) \quad k=1,\ldots,d. \tag{75}$$

The new object x_* is assigned to the class k, where kth rule provides the maximum score for x_*:

$$k=\text{argmax}\{s_*^1,\ldots,s_*^d\}, \quad \text{where} \quad s_*^t=f_t(x_*). \tag{76}$$

This method of d-class classification is not based on a clear statistical foundation. Another common heuristics called *one versus one* (OVO): it suggests to solve C_d^2 two-class classification problems separating all possible pairs of classes. To classify a new object x^*, one uses a voting scheme based on the obtained C_d^2 rules.

Here we implement the following multi-class classification procedure. For every k (where $k=1,\ldots,d$), we solve the corresponding OVR SVM problem, for which all the elements with the original label k are marked with $y=1$, while all the other elements are marked with $y=0$. Upon solving all these d problems, we can, for any given vector x and any class k, compute its score $s_k(x)$ provided by the kth SVM rule. After that, we merge the scores of these auxiliary SVM rules following the approach described in our paper [13].

We compared our synergy approach with the standard OVR approach for the data sets Vehicle, Waveform, and Cardiotocography from UCI Machine Learning Repository [12]. Training and test sets were selected randomly from these data sets; the number of elements in each are shown in Table 5; the table also shows the error rates achieved by OVR and synergy algorithm, along with relative performance gain obtained with our approach. The results confirm the viability of our framework.

Table 5. Synergy for Multi-Class Classificaion.

Data set	Classes	Features	Training	Test	OVR	Synergy	Gain
Vehicle	4	18	709	236	17.45%	14.15%	18.91%
Waveform	3	40	200	4800	20.10%	18.31%	8.90%
Cardiotocography	3	21	300	1826	15.83%	12.05%	23.87%

Thus, we showed that:

1. Scores $s=(s^1,\ldots,s^d)$ of several monotonic classifiers (for example, SVMs) that solve the same pattern recognition problem can be transformed into multi-dimensional monotonic conditional probability functions $P(y|s)$ (probability of class y given scores s).
2. There exists an effective algorithm for such transformation.
3. Classification rules obtained on the basis of constructed conditional probability functions significantly improve performance, especially in multi-class classification cases.

3.2 Data-Driven Approach

In this section, we focus on the standard general binary classification problem setting. In this problem, there is training set consisting of L samples, each being an N-dimensional vector that belongs to one of two classes, positive and negative, traditionally labeled as -1 and +1. For this setting, a decision rule has to be learned on the given set of L vectors, which can then be applied for classification of any arbitrary N-dimensional vector into one of two classes, -1 or +1 with minimum possible error rate (in other words, with minimum possible probability of misclassification, i.e., assignment of the wrong class label to the vector). SVM is the current best-in-class algorithm for solving this type of problem.

We explored the applicability of the current feature construction approach to a special area of discriminative learning (Figure 3) – Learning Using Privileged Information (LUPI), which essentially relies on two distinct classes of features (standard and privileged). The developed approach (in our papers [14] [15] [16] [17] [18]) of leveraging derived features was successfully carried over to the area of Learning using Privileged Information (LUPI) by using the regression mechanism of generating derived features (the one we originally proposed using for SVM) for approximating the privileged features in LUPI using standard ones.

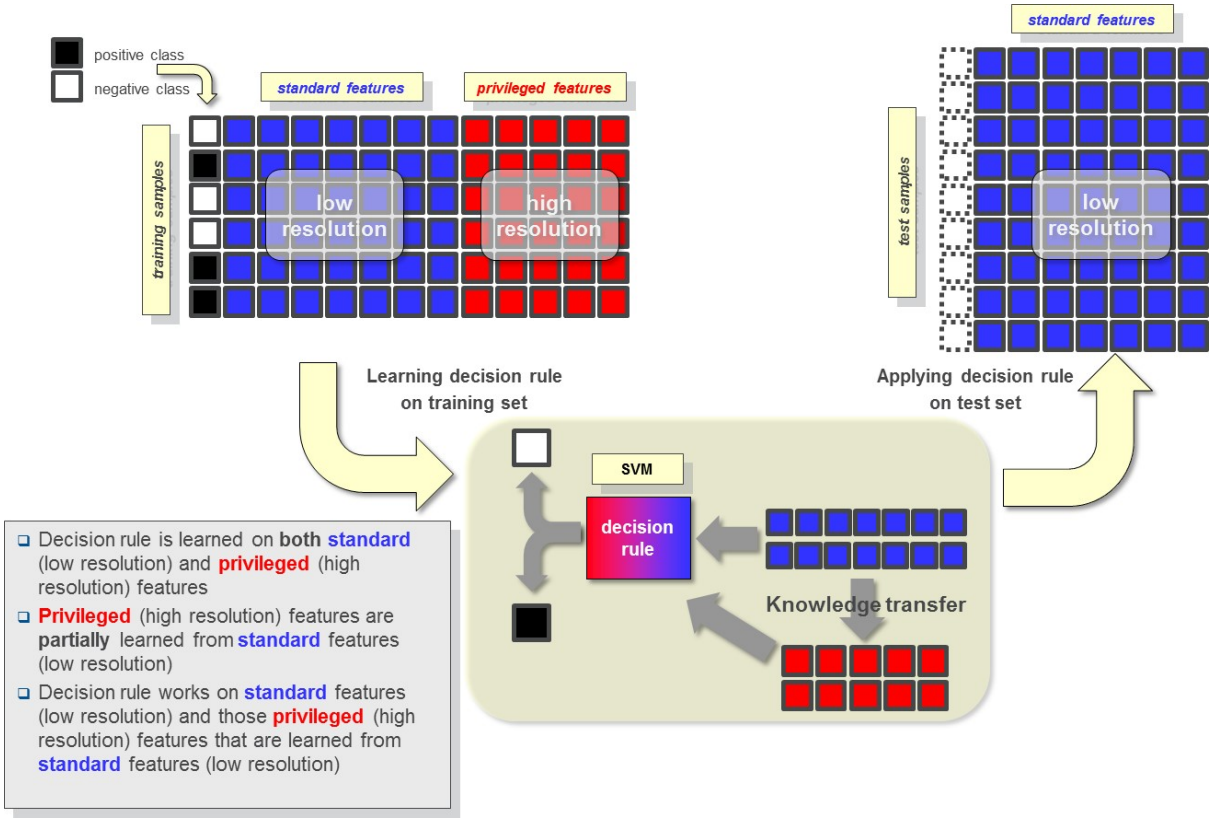

Figure 3. Novel Framework for Learning Using Privileged Information.

By using these approach, we were capable to capture some of the information contained in the privileged features and, by expressing it through standard features in the form of regression, we opened a new possibility of solving LUPI problems not just by controlling similarity between vectors (as was done before), but by information transfer from the space of privileged features to the space of standard ones. Even more importantly, this information transfer method essentially converts LUPI in a special form of SVM, thus completely resolving the main computational issue of scalability that was the key disadvantage of LUPI up until now. Indeed, current LUPI is only capable of handling about 300 training example in a reasonable time, whereas, with the new DILEGENT approach, we were able to process a sample size containing 2000 vectors (representing pixel data for image classification) without any computational problems.

In another example, using set of of pre-processed video snapshots of a terrain, one has to separate pictures with specific targets on it (class +1) from pictures where there are no such targets (class -1). The original videos were made using aerial cameras of different resolutions: a low resolution camera with wide view (capable to cover large areas quickly) and a high resolution camera with narrow view (covering smaller areas and thus unsuitable for fast coverage of terrain). The goal was to make judgements about presence or absence of targets using wide view camera that could quickly span large surface areas. The narrow view camera could be used during training phase for zooming in the areas where target presence was suspected, but it was not to be used during actual operation of the monitoring system, i.e., during test phase. Thus, the wide view camera with low resolution corresponds to standard information, whereas the narrow view camera with high resolution corresponds to privileged information.

Modern data analysis problems require construction of decision rules that operate in high dimensional spaces. Thus, in order to obtain good decision rules, one has to train learning algorithms using a huge number of

examples (tens or hundreds of thousands). According to the statistical learning theory [19] [20], which provides precise estimates of the convergence of learning algorithms, even the best state-of-the-art learning algorithms such as Support Vector Machines (SVM) will require plenty of training examples and computation time. At the same time, we know that humans can learn well from a small number of training examples. This gap in learning performance between humans and machines has been a persistent challenge from the very beginning of the computer era.

Learning Using Privileged Information (LUPI) was introduced first as an idea in [20] and subsequently realized in several versions of SVM+ algorithms, generalizing SVM with different degrees of implementation complexity [21] [22]. The idea of LUPI is to improve classification accuracy with small training datasets by fusing the training data with additional features, which are not present in the testing data. These features can come from an entirely different sensor modality. An example would be the utilization of medical records accompanying X-ray images of existing patients to better classify X-ray images of new patients who have not yet been diagnosed.

In the wide area aerial video exploitation scenario, which is the application considered by this contribution, the additional features come from high resolution videos or still imagery coincident with the training data. These additional data help reduce the number of data samples required to train accurate models of class distributions, which is important in dynamic environments.

As a development of LUPI, recent paper [9] introduced the concept of intelligent learning, based on the ideas of *knowledge transfer*, which allows to further improve classifier performance. Improved scalability is achieved by casting the problem within the SVM framework, for which multiple efficient algorithms are already implemented on most platforms; the improvement in flexibility is achieved by the ability to utilize a variety of methods within the knowledge transfer framework.

Next, we formulate, based on the general framework we developed in our papers [14] [15] [16] [17] [18], two specific types of knowledge transfer algorithms: privileged feature regression and privileged clustering. We further consider a combination of several knowledge transfer models in ensemble type learning. We compare the performance, measured in misclassification error rate and execution time, of both types of LUPI algorithms with the original SVM+ algorithm. We apply the algorithms to a wide area aerial video exploitation problem, where the privileged information represents more expensive and/or higher quality sensor information available for training data, but not for testing data. Using the Minor Area Motion Imagery (MAMI) dataset recently collected by Air Force Research Laboratory (Freeman, 2014), we demonstrate that knowledge transfer approach to LUPI provides consistently better performance than the original SVM+ approach. Using an ensemble of several knowledge transfer algorithms, the error rate is reduced by up to 25%. We also demonstrate a significant computational speedup, making LUPI algorithm as scalable as the standard SVM.

The traditional machine learning paradigm is formulated as follows. Given a set of training examples, and a parameterized collection of decision functions, find the function that approximates the unknown decision rule in the best possible way [9] [20]. Formally, for a binary classification problem, we are given L training vectors $X_1, X_2, \ldots, X_L \in R^N$, and the corresponding labels $y_1, y_2, \ldots, y_L \in \{-1, +1\}$.

In SVM approach, some kernel $K(X_i, X_j)$ is selected in the space R^N. Then, for positive penalty parameters $C_1, C_2, \ldots, C_L$, (usually equal to the same number C normalized by the ratio of positive or negative labels in the

training sample, respectively) a quadratic optimization problem is solved in order to find the values of parameters $\alpha_1,\alpha_2,\ldots,\alpha_L$ that maximize the following functional

$$\sum_i \alpha_i - \frac{1}{2}\sum_{ij} y_i y_j \alpha_i \alpha_j K(X_i, X_j) \tag{77}$$

subject to constraints

$$\sum_i \alpha_i y_i = 0,\ 0 \le \alpha_i \le C_i, i = 1, \ldots, L \tag{78}$$

The parameters $\alpha_1,\alpha_2,\ldots,\alpha_L$ are used to construct the decision function

$$f(z) = sgn\left(\sum_i \alpha_i y_i K(X_i, z) + B\right) \tag{79}$$

The offset B is computed for some training index j, such that $0 \le \alpha_j \le C_j$

$$B = y_j\left(1 - \sum_{ij} y_i y_j \alpha_i K(X_i, X_j)\right) \tag{80}$$

The parameters $C_1,C_2,\ldots,C_L$ and the kernel parameters of the SVM algorithm are usually optimized by executing cross-validation over some grid in the parameter space.

The LUPI paradigm can be described as learning with a teacher. In the classical model of learning described above, the teacher supplies the set of labels, and his/her role is trivial. In the LUPI model, the teacher supplies students with non-trivial additional information in various forms, such as images, explanations, metaphors, etc. This additional information is present only during the training stage (when the teacher is available), and will not be present during the test stage (when the teacher is not available).

Formally, LUPI approach is described as follows: at the training stage, we have L standard vectors

$$X_1, X_2, \ldots, X_L \in R^N, \tag{81}$$

and, corresponding to them, L privileged vectors

$$x_1, x_2, \ldots, x_L \in R^M \tag{82}$$

and L labels

$$y_1, y_2, \ldots, y_L \in \{-1, +1\}. \tag{83}$$

The decision rule has to operate only on the standard N-dimensional space R^N, as the test vectors belong to that space, and no privileged information will be available.

The original algorithm implementing the LUPI paradigm, SVM+, was designed as a generalization of the SVM algorithm [21] [22]. In SVM+, two kernels $K(X_i,X_j)$ and $k(x_i,x_j)$ are selected respectively in the standard space R^N and the privileged space R^M. Then, for a fixed positive structural parameters κ and γ and positive penalty parameters $C_1,C_2,\ldots,C_L$, SVM+ solves quadratic programming problem of finding the parameters $\alpha_1,\alpha_2,\ldots,\alpha_L$ and $\delta_1,\delta_2,\ldots,\delta_L$ that maximize the functional

$$\sum_i \alpha_i - \frac{1}{2}\sum_{ij} y_i y_j \alpha_i \alpha_j K(X_i, X_j) \tag{84}$$

$$-\frac{1}{2\gamma}\sum_{ij} y_i y_j(\alpha_i - \delta_i)(\alpha_j - \delta_j)k(x_i, x_j),$$

subject to constraints

$$\begin{gathered}\sum_i \alpha_i y_i = 0,\\ \sum_i \delta_i = \sum_i \alpha_i\,, \qquad 0 \le \alpha_i \le \kappa C_i, 0 \le \delta_i \le C_i,\\ i = 1, \dots, L\end{gathered} \tag{85}$$

There are other versions of SVM+ with similar performance characteristics. The version used in this report is based on [17]. The parameters are used to construct the decision function

$$f(z) = sgn(\textstyle\sum_i \alpha_i y_i K(X_i, z) + B) \tag{86}$$

where the offset B is computed for some index j, such that $0 \le \alpha_j \le C_j$ and $0 \le \delta_j \le C_j$:

$$B = y_j\left(1 - \sum_{ij} y_i y_j \alpha_i K(X_i, X_j) - \gamma \sum_{ij} y_i y_j(\alpha_i - \delta_i)k(x_i, x_j)\right) \tag{87}$$

The parameters $C_1, C_2, \dots, C_L$, κ, γ and kernel parameters of the SVM+ algorithm are optimized by grid search over the parameter space similar to the SVM algorithm.

With the growth of the number of training examples, the LUPI approach converges to the optimal solution much faster than the classical approaches. As a result, LUPI can require significantly fewer training examples than classical approaches to achieve the same level of performance. In some cases, LUPI can achieve the same performance by training with $\sqrt{L}$examples where the classical approach uses L examples – requiring, for example, only 320 examples instead of 100,000 examples required by the classical learning algorithms. LUPI approach has been already successfully applied to a diverse set of problems from multiple disciplines [23] [24] [25].

While delivering performance results that were previously impossible to achieve within the standard machine learning paradigm, the key problem with existing LUPI implementations, such as SVM+ [21] [22], was their limited scalability: since the core matrix in the quadratic programming implementation of SVM+ is poorly conditioned and SVM+ requires more parameters to tune, the practical limit of training sample size was about 200 examples (for larger samples, algorithms required days and weeks to converge). Although specially designed spline kernels [10] allowed increasing that sample size to 300-350 examples, the scalability problem remained the main obstacle for much wider applications of LUPI.

To reiterate, there are several factors that make SVM+ more computationally expensive:

i. The quadratic optimization problem associated with SVM+ is twice the size of the one associated with SVM and is more complex, and so it takes more time to solve;
ii. SVM+ has four free parameters - twice the number of free parameters in the SVM algorithm. Tuning four parameters requires more computations.
iii. The core matrix in the quadratic optimization problem is usually ill-conditioned, which significantly slows down the quadratic optimization process.

All these factors, taken together, define the practical limit of training sample size for the SVM+ algorithm at about 200-300 examples.

Recently, the concept of LUPI was theoretically expanded to include more general knowledge transfer mechanisms [14] [15] [16] allowing to transfer knowledge from the space of privileged information (space of Teacher's explanations) to the space where decision rule is constructed.

We now formulate a specific algorithm [17] implementing knowledge transfer via regression of privileged features. Here, we consider two versions of the privileged regression approach: one that uses linear ridge regression for approximating privileged features, and another one that uses kernel regression with RBF functions. In both versions, regularization parameters (and for non-linear regression, the Gaussian parameter of RBF function) are selected using 2-fold cross validation.

Consider a training set consisting of L standard vectors $X_1, X_2, \dots, X_L \in R^N$ and corresponding to them L privileged vectors $x_1, x_2, \dots, x_L \in R^M$. We assume that each of the vectors X_i is labeled with $y_i \in \{-1, +1\}$. In matrix form (where the subscripts denote vector indices and superscripts denote vector elements), the training set has the form

$$\begin{pmatrix} y_1 & X_1^1 & X_1^2 & \dots & X_1^N & x_1^1 & x_1^2 & \dots & x_1^M \\ y_2 & X_2^1 & X_2^2 & \dots & X_2^N & x_2^1 & x_2^2 & \dots & x_2^M \\ \dots & \dots & \dots & \dots & & & \dots & & \\ y_L & X_L^1 & X_L^2 & \dots & X_L^N & x_L^1 & x_L^2 & \dots & x_L^M \end{pmatrix} \tag{88}$$

We now describe the way knowledge transfer LUPI algorithm works for this training set if the parameters of the desired SVM classification decision rule are already known; if they are not known, the parameter search is executed as described further.

Train-1. For each $j = 1,2, \dots, M$, do the following. Using N-dimensional vectors $X_1, X_2, \dots, X_L$ as explanatory variables and corresponding scalar values $x_1^j, x_2^j, \dots, x_L^j$ as response variables, construct a (linear or nonlinear) regression function φ_j so that

$$\begin{gathered} \varphi_j(X_1^1, X_1^2, \dots, X_1^N) = z_1^j \approx x_1^j, \\ \varphi_j(X_2^1, X_2^2, \dots, X_2^N) = z_2^j \approx x_2^j, \\ \dots \quad \dots \quad \dots \\ \varphi_j(X_L^1, X_L^2, \dots, X_L^N) = z_L^j \approx x_L^j \end{gathered} \tag{89}$$

Train-2. Use the values z_i^j (where $j = 1,2, \dots, M$ and $= 1,2, \dots, L$) constructed in the previous step to augment the vectors $X_1, X_2, \dots, X_L$ from N-dimensional space R^N to form vectors $Z_1, Z_2, \dots, Z_L$ from $(N + M)$-dimensional space R^{N+M}; these vectors have the matrix form

$$\begin{gathered} Z_1^T = (X_1^1 \quad X_1^2 \quad \dots \quad X_1^N \quad z_1^1 \quad z_1^2 \quad \dots \quad z_1^M), \\ Z_2^T = (X_2^1 \quad X_2^2 \quad \dots \quad X_2^N \quad z_2^1 \quad z_2^2 \quad \dots \quad z_2^M), \\ \dots \quad \dots \quad \dots \\ Z_L^T = (X_L^1 \quad X_L^2 \quad \dots \quad X_L^N \quad z_L^1 \quad z_L^2 \quad \dots \quad z_L^M). \end{gathered} \tag{90}$$

As a result, for each $i = 1,2, \dots, L$, the first N elements of vector Z_i constitute standard vector X_i, while

the last M elements of vector Z_i constitute regression-based approximations ($z_i^1 \quad z_i^2 \quad \dots \quad z_i^M$) to privileged vector x_i.

Train-3. Create the new (augmented) training set with vectors $Z_1, Z_2, \dots, Z_L$ and corresponding labels $y, y_2, \dots, y_L$; in matrix form, the augmented training set has the form

$$\begin{pmatrix} y_1 & X_1^1 & X_1^2 & \dots & X_1^N & z_1^1 & z_1^2 & \dots & z_1^M \\ y_2 & X_2^1 & X_2^2 & \dots & X_2^N & z_2^1 & z_2^2 & \dots & z_2^M \\ \dots & \dots & \dots & \dots & & \dots & & & \\ y_L & X_L^1 & X_L^2 & \dots & X_L^N & z_L^1 & z_L^2 & \dots & z_L^M \end{pmatrix}. \tag{91}$$

Train-4. Train an SVM algorithm with known parameters on vectors $Z_1, Z_2, \dots, Z_L$ from $(N+M)$-dimensional space R^{N+M} and construct the corresponding classification decision function F, which, when applied to any $(N+M)$-dimensional vector Z, produces the classification output $Y = F(\mathrm{Z})$, where $Y \in \{-1, +1\}$. Otherwise, if the parameters are unknown, carry out SVM parameter search as described further in sub-section C.

The designed classification decision algorithm F can now be applied to any standard vector X from N-dimensional space R^N in following manner.

Test-1. Using already constructed (during training) M regressions $\varphi_1, \dots, \varphi_M$, construct M scalar values

$$z^1 = \varphi_1(X), z^2 = \varphi_2(X), \dots, z^M = \varphi_M(X) \tag{92}$$

Test-2. Construct $(N+M)$-dimensional vector Z by concatenating these M scalar values with N-dimensional vector X:

$$Z = (X^1 \quad X^2 \quad \dots \quad X^N \quad z^1 \quad z^2 \quad \dots \quad z^M) \tag{93}$$

Test-3. Apply classification decision algorithm F to the constructed $(N+M)$-dimensional vector Z and obtain the classification label $Y = F(\mathrm{Z})$, where $Y \in \{-1, +1\}$. This label Y is the desired classification of standard N-dimensional vector X.

In other words, the designed (N+M)-dimensional decision rule is used for any test N-dimensional test vector Z in three steps: (1) from already constructed (at training stage) M multivariate regressions, compute M approximations to missing privileged features, (2) concatenate the N-dimensional test vector Z with these M approximations, and (3) apply the decision rule to the resulting (N+M)-dimensional augmented test vector. The full feature space thus combines the original features and the new features from a domain knowledge (similar to [1] [26] [27], where domain knowledge is represented by the privileged feature space.

As already mentioned, the training procedure described above is applicable if the parameters of the desired SVM classification decision rule are already known (for instance, for SVM with RBF kernel there are SVM penalty parameter C and Gaussian parameter γ). If they are not known (which is usually the case), they are selected using grid search over a pre-defined set of parameter vectors q. In case of SVM with RBF kernel, this set is 2-dimensional. In general, we assume that this set consists of P vectors $\{q_1, q_2, \dots, q_P\}$. The search is performed in the following way (for simplicity, we describe it in the case of 6-fold cross-validation).

Param-1. The training set X is randomly partitioned into six subsets $X_1, X_2, \dots, X_6$ of approximately equal size.

Param-2. For each $i = 1,2 \dots, P$, the cross-validation error rate $E_i$of the algorithm with parameter vector q_i is computed in the following way .

Param-2-1. For each $k = 1,2 \ldots, 6$, the following operation is executed.

Param-2-1-1. Auxiliary sets X_{train} and X_{test} are formed: X_{test} is the k^{th} of sets $X_1, X_2, \ldots, X_6$, and X_{train} is the union of the other five subsets.

Param-2-1-2. Apply steps Train-1, Train-2, Train-3, Train-4 to X_{train} to compute the classification decision function F_{ik}.

Param-2-1-3. Apply the classification decision function F_{ik} to X_{test} according to Test-1, Test-2, Test-3 and compute the resulting classification error rate E_{ik}.

Param-2-2. Compute the average E_i of six error rates $E_{i1}, E_{i2}, \ldots, E_{i6}$.

Param-3. Among all $i = 1,2 \ldots, P$, select the parameter vector q_i corresponding to the smallest error rate E_i.

In terms of scalability, it is clear that both linear and nonlinear versions of LUPI knowledge transfer algorithms avoid the main problem of SVM+: while the additional step of calculating *M* multivariate regressions takes some time, the regression is performed only once during the whole grid search, and, most importantly, the augmented training data are then processed by standard scalable SVM implementations.

In terms of performance, it is important to gauge it properly. Assuming that the quality of the designed LUPI solution is *A* (i.e., its error rate is *A*%), we can compare it with the quality of two standard learning solutions:

(1) Only standard features are used, i.e., SVM in *N*-dimensional space.
(2) Both standard and privileged features are used, i.e., SVM on (*N*+*M*)-dimensional space.

The solution (1) corresponds to the pre-LUPI situation when privileged features may exist, but they are discarded since there is no mechanism to take them into account. The solution (2) is the ideal situation, when all privileged features do not disappear during the test phase, but instead remain as standard features. Assuming that the solution (1) has quality *B* (i.e., its error rate is *B*%), while the solution (2) has quality *C* (i.e., its error rate is *C*%), we can generally expect that *C*<*A*<*B*. Indeed, *C* should be the smallest one among the three, since all privileged features are actually standard, so the power of SVM solution can be used to its fullest. Also, *B* should be the largest one among the three since it corresponds to pre-LUPI situation of classical learning, where all privileged features are ignored. So the quality of our LUPI solution, being "sandwiched" between *C* and *B*, can be naturally evaluated by measuring how much progress LUPI could make within the performance gap *B*–*C*. Thus, the metric (*B*–*A*)/(*B*–*C*) can be used to assess the relative quality of the LUPI solution.

Note that if gap *B*–*C* is small, it means that privileged information is not particularly relevant (its knowledge in solution (2) does not improve much the quality of solution (1)), then it's probably hopeless to apply LUPI anyway – there is no space for improvement for that. Given our experience, it is probably safe to start looking for LUPI solution if the gap *B*–*C* is at least 1.5-2 times larger than *C*. The improvement metric (*B*–*A*)/(*B*–*C*) of about 20-30% or more would then constitute success of LUPI approach.

The MAMI classification dataset [28] is based on mover extraction from airborne wide area imagery collected by AFRL. The goal of the classifier is to improve dismount detection in low-resolution motion imagery. As described, for instance, in [29], dismount tracking is the concept of tracking a person either by direct observation or indirectly by inference, such as determining where the person was when exiting direct view. Dismount tracking is an important security application in that a nominated person can be tracked through various activities to predict and mitigate harmful actions, establish intent, and determine social group association [29]. In its native resolution, the MAMI imagery has dismounts of 20-pixel height, while imagery

downsampled to 1/8th resolution has 2-3 pixel high dismounts. Motion imagery is generated at about 15 Hz. The data were collected by multiple cameras with different resolutions.

We used test data from June 26, 2013. These images are centered on a grassy area where a picnic is taking place. There are dozens of people milling, walking, running, sitting, and playing volleyball and other games. There are also some scripted dismount activities and parked and moving vehicles. Most dismounts are taken from the picnic area which was in Camera 4's Field of View (FOV), see Figure 4. Some dismounts are also in the parking lots, and near roads and paths. Most false alarms are from outside the picnic area, especially in the Camera 3 FOV, which points below Camera 4. False alarms were caused by parallax, trees, poles, buildings, vehicles, reflections, image noise, and registration errors.

To test the LUPI algorithms, we used the images at native and reduced resolutions. Low-resolution 1/N*th* images were formed by averaging the intensities of NxN blocks of pixels into one pixel and then interpolating the reduced image back to the original resolution. We grouped images into sequences, each representing 10-15 seconds of data. For each sequence, we designated a target area, where we expected to find multiple dismounts, and a false alarm area, where we expect to see few or no dismounts. We registered the images in a sequence and extracted tracklets.

We built a background model and used differences between the images and the background models to detect mover candidates. Detections from the frames of sequences are linked together to form tracks. We tuned the tracker at each resolution to have 70-90% Probability of Detection (PD) on the true dismount samples. We describe the detected mover candidates by a set of features and use LUPI techniques to separate the true movers from false alarms.

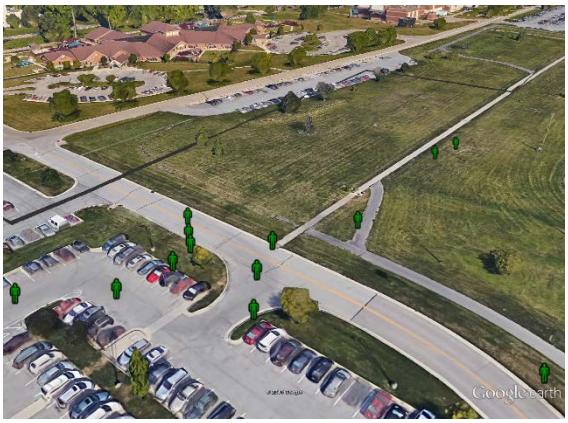

Figure 4. Dismounts Detected in MAMI Data.

We built the following features for each detected mover: 45 object-level features (including blob, kinematic, track, fraction of frames in which this mover was detected, mean of length measurements for this mover, divided by standard deviation, eccentricity of best fit ellipse, standard deviation of orientation measurements over all frames, median of the mean pixel values for each observation of this mover, etc.), 20 activity-level features (turns, accelerations, starts, stops, etc.), 64 SURF features, and 4 Gradient features, for a total of 113 features.

We selected 14 of these features (such as area, eccentricity) as reasonably robust to be used as the standard features to be used in the design classifier, while we use all features at the full resolution as the privileged

space. We constructed the dataset consisting of 856 objects described by 127 features – 14 standard and 113 privileged. 567 objects were labeled as positive class (dismounts), and 289 – as negative (false alarms). We extracted 567 positive and 289 negative samples from the MAMI data. In each experiment, we used a set of 30 randomly sampled sets of training size L= 80, 160, 320, 640, from which we formed balanced training sets with equal number of objects N_T=27, 54, 108, 216 of each of 2 classes to construct a classifier. For each training set, we used K random partitions in 80% training, 20% tuning to optimize parameters of each of the algorithms. Remaining objects that were not selected in training set form holdout set. So we have 30 pairs of training and holdout sets.

We optimized the baseline SVM over the grid C = 0.1,1,10, $\gamma = 2^{-6}, 2^{-4}, 2^{-2},\ 1,\ 2^{2},\ 2^{4},\ 2^{6}$, using K = 10; SVM regression over the grid C = 0.1,1,10, $\gamma = 0.1, 1, 10$; LPC Linear over 1 to 5 clusters separately for positive and negative classes and C=0.1,1,10. LPC Gauss used fixed parameters: 3 and 2 clusters respectively for positive and negative classes, C=1, $\gamma = \left(\frac{2}{N_T(N_T-1)}\sum_{i<j}\left\|X_i - X_j\right\|^2\right)^{1/2}$, where X_i, X_jobjects from training set. Parameters for each training set were optimized for the case of N_T=27 and same values used for large training samples.

We tested performance as weighted average error on the 30 holdout sets. The obtained results suggest the following conclusions:

i. LUPI-regression algorithm consistently outperforms the baseline;
ii. LPC algorithm on average outperforms the baseline;
iii. Scalability problem of LUPI SVM+ was confirmed again for samples larger than 200-300 elements, as we can see from the entry 23391 for L=640 in Table I, which corresponds to 16 days of optimization;

These observations demonstrate that general knowledge transfer approach in LUPI paradigm can be implemented in scalable algorithms successfully leveraging privileged information by various means. The results also demonstrate the value of fusing multiple mechanisms of knowledge transfer within the LUPI paradigm. Thus we have formulated and implemented two new closely related classification algorithms within the recently introduced approach of knowledge transfer in LUPI. Both algorithms successfully resolve the scalability problem of previous LUPI approaches and allow for diverse and scalable leveraging of privileged information in classification problems. We verified the proposed approaches using dismount detection problem in AFRL MAMI airborne data and demonstrate that efficacy of the proposed algorithms, both in terms of their performance and scalability.

In order to explore privileged information and knowledge transfer in more detail, we considered the following approach.

Let us suppose that Intelligent Teacher has some knowledge about the solution of a specific pattern recognition problem and would like to transfer this knowledge to Student. For example, Teacher can reliably recognize cancer in biopsy images (in a pixel space *X*) and would like to transfer this skill to Student.

Formally, this means that Teacher has some function $y=f_0(x)$ that distinguishes cancer ($f_0(x)=+1$ for cancer and $f_0(x)=-1$ for non-cancer) in the pixel space *X*. Unfortunately, Teacher does not know this function explicitly (it only exists as a neural net in Teacher's brain), so how can Teacher transfer this construction to Student? Below, we describe a possible mechanism for solving this problem; we call this mechanism *knowledge transfer*.

Suppose that Teacher believes in some theoretical model on which the knowledge of Teacher is based. For cancer model, he or she believes that it is a result of uncontrolled multiplication of the cancer cells (cells of type B) which replace normal cells (cells of type A). Looking at a biopsy image, Teacher tries to generate privileged information that reflects his or her belief in development of such a process; Teacher can describe the image as:

Aggressive proliferation of cells of type B into cells of type A.

If there are no signs of cancer activity, Teacher may use the description

Absence of any dynamics in the standard picture.

In uncertain cases, Teacher may write

There exist small clusters of abnormal cells of unclear origin.

In other words, Teacher uses a specialized language that is appropriate for description x_i^* of cancer development employing the model he believes in. Using this language, Teacher supplies Student with privileged information x_i^* for the image x_i by generating training triplets

$$(x_1,x_1^*,y_1),\ldots,(x_\ell,x_\ell^*,y_\ell). \tag{94}$$

The first two elements of these triplets are descriptions of an image in two languages: in language X (vectors x_i in pixel space), and in language X^* (vectors x_i^* in the space of privileged information), developed for Teacher's understanding of cancer model.

Note that the language of pixel space is universal (it can be used for description of many different visual objects; for example, in the pixel space, one can distinguish between male and female faces), while the language used for describing privileged information is very specialized: it reflects just a model of cancer development. This has an important consequence: the set of admissible functions in the general space X has to be rich (has large VC dimension), while the set of admissible functions in the specialized space X^* may be not rich (has small VC dimension).

One can consider two related pattern recognition problems using triplets:

1. The problem of constructing a rule $y=f(x)$ for classification of biopsy in the pixel space X using data
$$(x_1,y_1),\ldots,(x_\ell,y_\ell). \tag{95}$$
2. The problem of constructing a rule $y=f^*(x^*)$ for classification of biopsy in the space X^* using data
$$(x_1^*,y_1),\ldots,(x_\ell^*,y_\ell). \tag{96}$$

Suppose that language X^* is so good that it allows to create a rule $y=f_\ell^*(x^*)$ that classifies vectors x^* corresponding to vectors x with higher accuracy.

Since the VC dimension of the admissible rules in the specialized space X^* is much smaller than the VC

dimension of the admissible rules in the universal space X and since the number of examples ℓ is the same in both cases, the bounds on error rate for the rule $y=f^*_\ell(x^*)$ in X^* will be better (according to VC theory, the guaranteed bound on accuracy of the chosen rule depends only on two factors: frequency of errors on the training set and VC dimension of the admissible set of functions.) than those for the rule $y=f_\ell(x)$ in X. That is, generally speaking, the classification rule $y=f^*_\ell(x^*)$ will be more accurate than classification rule $y=f_\ell(x)$.

As a result, the following question of "knowledge transfer" arises: how one can use the knowledge of the rule $y=f^*_\ell(x^*)$ in space X^* to improve the accuracy of the desired rule $y=f_\ell(x)$ in space X? We now address this question when both described problems are solved with neural networks.

Consider three elements of knowledge representation used in Artificial Intelligence:

1. Fundamental elements of knowledge.
2. Frames (fragments) of the knowledge.
3. Structural connections of the frames (fragments) in the knowledge.

We call the *fundamental elements of the knowledge* a limited number of elements (functions) in X^* that can approximate well the classification rule $y=f^*_\ell(x^*)$; then knowledge transfer is about approximation of those fundamental elements. We now illustrate this concept for SVMs and neural networks.

In order to describe methods of knowledge transfer for SVM, consider the following three-level structure:

1. Level I^X: the input vectors $x=(x^1,\ldots,x^n)\in X$.
2. Level I^Z: the result of transformation of the vectors x into vectors $z=(K(x_1,x),\ldots,K(x_\ell,x))\in Z$, where K is the kernel function for SVM.
3. Level I^Y: the linear threshold indicator function [1] $y=\Theta(a^T z(x)-b)$ in space Z.

Thus the structures of SVM rules in spaces X and X^* can be described as

$$X: \left(I^X \rightarrow I^Z \rightarrow I^Y\right) \quad \text{and} \quad X^*: \left(I^{X^*} \rightarrow I^{Z^*} \rightarrow I^{Y^*}\right). \tag{97}$$

To transfer the knowledge about the rule

$$y=f(x^*,\alpha^*_\ell)-b^* = \sum_{i=1}^{\ell} \alpha^*_i K^*(x^*_i,x^*)-b^* \tag{98}$$

in space X^* to the rule

[1] Neural Network with one hidden layer has the same structure; as SVM, it is a *universal* learning machine.

$$y=f(x,\alpha_\ell)-b=\sum_{i=1}^{\ell}\alpha_i K(x_i,x)-b \tag{99}$$

obtained in space X, one can use several strategies. Below we consider three of them.

1. **A-mapping of privileged information: $\mathbf{X^*\rightarrow X}$**. In this scheme, the goal is to transfer information that exists in level I^{X^*} of SVM in space X^* to level I^X of SVM in space X. In order to do this, one maps vectors $x\in X$ into vectors $x^*\in X^*$ by transforming space X obtaining vectors $\overline{x}=Ax$ and then constructs SVM in the transform space $\overline{X}$. Scheme **(A)** of information transfer can be thus described as

$$\textbf{(A):}\quad \left(I^{X^*}\rightarrow I^X\right)\rightarrow I^Z\rightarrow I^Y \tag{100}$$

In this scheme, in order to find the transformation Ax of vectors $x=(x^1,\ldots,x^n)^T\in X$ into vectors $Ax=(\varphi_1(x),\ldots,\varphi_m(x))^T$ that minimizes the functional

$$R(A)=\min_A \int |x^*-Ax|^2 p(x^*,x)dx^*dx, \tag{101}$$

we look for the minimum

$$R(\varphi)=\sum_{k=1}^{n}\min_{\varphi_k}\int (x^{*k}-\varphi_k(x))^2 p(x^{*k},x)dx^{*k}dx, \tag{102}$$

where $p(x^{*k}|x)$ is the marginal conditional probability of coordinate x^{*k} given vector x, and m functions $\varphi_k(x)$ are defined by m regressions

$$\varphi_k(x)=\int x^{*k}p(x^{*k}|x)dx^{*k},\quad k=1,\ldots,m. \tag{103}$$

We construct approximations to functions $\varphi_k(x)$ by solving m regression estimation problems based on data

$$(x_1^{*k},x_1),\ldots,(x_\ell^{*k},x_\ell)),\quad k=1,\ldots,m. \tag{104}$$

In order to find these approximations, we Structural Risk Minimization principle [19] in the set of functions that belong to the Reproducing Kernel Hilbert Space (RKHS) associated with some kernel, that is, by minimizing the regularized functional

$$R(\varphi_k)=\min_{\varphi_k}\sum_{i=1}^{\ell}(x_i^{*k}-\varphi_k(x_i))^2+\gamma\langle\varphi_k(x),\varphi_k(x)\rangle,\quad k=1,\ldots,m. \tag{105}$$

The obtained approximations to the regressions $\varphi_k(x)$ define our transformation. In this scheme, we first transform the input space $\overline{X}=AX$ and then train SVM in the transformed space.

2. **B-mapping of privileged information: $\mathbf{Z^*\rightarrow X}$**. In this scheme, the goal is to transfer information that exists in level I^{Z^*} of SVM in space X^* to level I^X of SVM in space X. In order do this, one maps vectors $x\in X$ to vectors $z^*\in Z^*$ by transforming space X and obtaining vectors $\overline{x}=Bx\in\overline{X}$ and then

constructs SVM in the transformed input space. Scheme **(B)** of information transfer can be thus described as

$$\textbf{(B):} \quad \left(I^{X^*} \to I^{Z^*}\right), \to I^X \to I^Z \to I^Y \tag{106}$$

or, in its simplified form, as

$$\textbf{(B'):} \quad \left(I^{X^*} \to I^{Z^*}\right) \to I^X \to I^Y. \tag{107}$$

The transformation of input space in this scheme is based on solving the following t regression estimation problems (t is the dimension of vector

$$z^* = (K(x_1^*, x^*), \ldots, K(x_t^*, x^*)^T) \quad , \tag{108}$$

i.e., the number of support vectors in SVM solution for space X^*): given data

$$(K(x_k^*, x_1^*), x_1), \ldots, (K(x_k^*, x_\ell^*), x_\ell), \quad k=1,\ldots,t, \tag{109}$$

find the regression functions

$$\varphi_k(x) = \int K(x_k^*, x^*) p(x^*|x) dx^*, \quad k=1,\ldots,t. \tag{110}$$

As already desrcibed above for A-mapping, one can find such approximation in the RKHS associated with some kernel function. The obtained approximations $\varphi_1(x),\ldots,\varphi_m(x)$ define our transformation: in general scheme, we construct SVM rule in the transformed space; in simplified scheme, we construct linear SVM rule in the transformed space $\overline{X}$.

3. **C-mapping of privileged information: $\mathbf{I^{Z^*} \to I^Z}$.** In this scheme, the goal is to transfer information that exists in the level I^{Z^*} of SVM in space X^* to the level I^Z of SVM in space X. In order to do this, one maps t-dimensional vectors $z \in Z$ (t is the number of support vectors of the SVM rule obtained in space X) into t^*-dimensional vectors $z^* \in Z^*$ (t^* is the number of support vectors of the SVM rule obtained in space X^*) constructing vectors of the form $\overline{z} = Cz \in \overline{Z}$. Every coordinate k in Z space defines similarity $K(x_k, x)$ between support vector x_k and vector $x \in X$, while every coordinate k^* in Z^* space defines similarity $K^*(x_k^*, x^*)$ between support vector x_k^* and vector $x^* \in X^*$, where x and x^* are connected through $p(x^*|x)$. Scheme **(C)** of information transfer can be described as

$$\textbf{(C):} \quad \left(I^{X^*} \to I^{Z^*}\right) \to \left(I^X \to I^Z\right) \to I^Y. \tag{111}$$

Our goal is to approximate the similarity function $K^*(x_k^*, x^*), k=1,\ldots,t^*$ between support vector x_k^* of SVM solution in space X^* and vector $x^* \in X^*$ using t similarity functions $K(x_1, x),\ldots,K(x_t, x)$ defined by SVM solution in space X for the pairs (x, x^*) generated by $p(x^*|x)$. Let $x_1,\ldots,x_t$ be the support vectors of SVM solution in space X and let $x_1^*,\ldots,x_{t^*}^*$ be the support vectors of SVM solution in space X^*, where t and t^* are the numbers of support vectors in SVM solutions obtained in spaces X and X^*, respectively. The SVM rule of the space X has the form

$$f(x,\alpha)=\sum_{i=1}^{t}\alpha_i K(x_i,x)+b, \tag{112}$$

and SVM rule in space X^* has the form

$$f^*(x^*,\alpha^*)=\sum_{i=1}^{t^*}\alpha_i^* K^*(x_i^*,x^*)+b^*. \tag{113}$$

In order to achieve our goal, we approximate the functions $K^*(x_k^*,x^*),k=1,\ldots,t^*$ with the regression functions

$$\varphi_k(x)=\int K(x_k^*,x^*)p(x^*|x)dx^*,\quad k=1,\ldots,t^*. \tag{114}$$

For each $k=1,\ldots,t^*$, we construct the approximation to $\varphi_k(x)$ by using the data

$$(K^*(x_k^*,x_1^*),z_1),\ldots,(K^*(x_k^*,x_\ell^*),z_\ell),\quad k=1,\ldots,t^*, \tag{115}$$

where $z_i=\left(K(x_1,x_i),\ldots,K(x_t,x_i)\ \in Z\right)$ is t-dimensional vector. Let space $\overline{Z}=(\varphi_1(x),\ldots,\varphi_t(x))$ be the result of transformation of space Z. After that, we construct linear SVM in space $\overline{Z}$.

One can construct many different schemes of knowledge transformation from space X^* to space X (as well as schemes of combining knowledge existing in both spaces) based on described approaches.

In particular, in all three described mappings A–C, one may also concatenate the constructed knowledge transferred features with those already available from space X and solve SVM on this augmented set; constructed knowledge transferred features could be subject to feature selection in order to improve the classification performance; for C-mapping, one could construct regression functions only to those functions $K^*(x_i^*,x^*)$ that correspond to "significant" weights α_i; if linear regression functions are used for C-mapping, their positive versions could be explored as more relevant, etc. Note that *C*-mapping requires executing two versions of SVM: one for standard space, and one for privileged one.

Knowledge transfer in Neural Networks is analogous to the one used for knowledge transfer in SVMs. As in the case of SVM described above, one constructs and trains two neural networks: one network in space X and another network in space X^*. To simplify the notations, we assume that both networks have the same architecture containing s layers. Let input vector $x\in X$ define the first layer $I^X(0)$ of neural network in space X; this vector is transferred into vector $z^1\in Z(1)$ in the next layer of the trained network, and layers $I^Z(k)$, $k=2,....,s-1$ provide subsequent transformations $z^k\in Z(k)$. As in SVM, the last layer is the linear indicator function $y=\Theta((a^s,z^s)-b)$ (or its sigmoid approximation). The structure of Neural Network in space X is

$$X:\ \ I^Z(0)\longrightarrow I^Z(1)\longrightarrow\cdots\longrightarrow I^Y. \tag{116}$$

and the structure of Neural Network in space X^* is

$$X^*:\ I^{Z^*}(0)\longrightarrow I^{Z^*}(1)\longrightarrow\cdots\longrightarrow I^Y. \tag{117}$$

The *simple scheme* of knowledge transfer from network in space X^* to network in space X can be described as follows: information accumulated into first k layers of network trained in space X^* is transferred into m-th layer of network in space X:

$$\left(I^{Z^*}(0)\rightarrow\cdots\rightarrow I^{Z^*}(k)\right)\rightarrow\left(I^{Z}(0)\rightarrow\cdots\rightarrow I^{Z}(m)\right)\rightarrow I(m+1)\rightarrow\cdots\rightarrow I^{Y}, \tag{118}$$

which forms the operator $\overline{z(k)}=Az(m)$ that transforms vectors $z(m)$ from neural network in space X into vectors $z^*(k)$ of neural network in space X^*.

The new neural network contains three parts:

1. The first part of the network contains first m layers of trained network in space X; we denote it $N(0,m)$. This network performs transformation $z(m)=N(0,m)x(0)$.
2. The second part of the network contains operator A that transforms vectors $z(m)$ in vectors $\overline{z(k)}=Az(m)$.
3. The third part of the networks is the part of the network in space X^* starting from level $(k+1)$, free parameters of which have to be learned; we denote it $N^*(k,s)$. Vectors $\overline{z(k)}$ are the input of this part of network, and classifiers are the output.

The scheme of such combined networks is

$$A\{N(0,m)\longrightarrow\}N^*(k,s), \tag{119}$$

where $N(0,m)$ is fixed (does not have free parameters), while $N^*(k,s)$ contains free parameters. Therefore operator A transforms knowledge about neural network in X^*.

In order to find this operator based on two trained networks, one uses the same techniques of regression estimation as in the case of SVM. Let $z^*(k)=(z^{*1}(k)....,z^{*s^*}(k))$ be vectors produced on the level $I^{Z^*}(k)$ by the network trained in space X^*, and let $z(m)=(z^1(m)....,z^s(m))$ be vectors produced on the level $I^Z(k)$ by the network trained in space X.

Consider pairs

$$(x_1,x_1^*),\ldots,(x_\ell.x_\ell^*) \tag{120}$$

from the training triplets. Let

$$z_1(m),\ldots,z_\ell(m) \tag{121}$$

be vectors produced by m-th layer of neural networks corresponding to vectors x and let

$$z_1^*(k),\ldots,z_\ell^*(k) \tag{122}$$

be vectors

$$z_i^*(k)=(z_1^{*1}(k),\ldots,z^{*s^*}(k)) \tag{123}$$

produced by k-th layer of neural networks corresponding to vectors x^*. In order to construct mapping operator A as in SVM case, we estimate s^* regression functions $x^{*t}(k)=\varphi_t(x(m))$ using data

$$(x_1^{*t},x_1(m)),\ldots,(x_\ell^{*t},x_\ell(m)),\quad t=1,\ldots,s^*. \tag{124}$$

Therefore operator A transforms vectors $x(m)$ into vectors

$$Ax(m)=(\varphi_1(x(m),\ldots,\varphi_{s^*}(x(m)). \tag{125}$$

For neural network that contains more than one hidden layer, one can transfer knowledge from network in X^* using more than one operator A_j, $j=1,\ldots,p$ by sequentially constructing several transformations between different layers of network in X^*.

Thus we described three key approaches for mapping of privileged information for knowledge transfer. In this section, we present scalable algorithms for one of them, namely A-mapping, based on multivariate regressions of privileged features as functions of decision variables; we also illustrate the algorithms' performance and their properties on several examples.

In order to illustrate this version of knowledge transfer LUPI, we explored the synthetic dataset derived from dataset "Parkinsons" in UCI Machine Learning Repository. Since none of 22 features of "Parkinsons" dataset is privileged, we created several artificial scenarios emulating the presence of privileged information in that dataset. Specifically, we ordered "Parkinsons" features according to the values of their mutual information (with first features having the lowest mutual information, while the last features having the largest one). Then, for several values of parameter k, we treated the last k features as privileged ones, while first $22-k$ features being treated as decision ones. Since our ordering was based on mutual information, these experiments corresponded to privileged spaces of various dimensions and various relevance levels for classification. For each considered value of k, we generated 20 pairs of training and test subsets, containing, respectively 75% and 25% of elements of the "Parkinsons" dataset. For each of these pairs, we considered the following four types of classification scenarios for both SVM (with RBF kernel) and ANN algorithms:

1. SVM and ANN on $22-k$ decision features;
2. Knowledge transfer LUPI (linear) based on constructing k multiple linear regressions from $22-k$ decision features to each of k privileged ones, replacing the corresponding values in privileged vectors with their regressed approximations, and then training SVM and ANN on the augmented dataset consisting of 22 features;
3. Knowledge transfer LUPI (non-linear) based on constructing k non-linear (in the class of RBF functions) regressions from $22-k$ decision features to each of k privileged ones, replacing the corresponding values in privileged vectors with their regressed approximations, and then training SVM and ANN on the augmented dataset consisting of 22 features;
4. SVM and ANN on all 22 features.

For each scenario, the algorithms were trained in the following way:
SVM. Two parameters for RBF kernels, namely SVM penalty parameter C and RBF kernel parameter γ, were selected using 6-fold cross-validation error rate over the two-dimensional grid of both parameters C and γ. In that grid, $\log_2(C)$ ranged of from -5 to $+5$ with step 0.5, and $\log_2(\gamma)$ ranged $+6$ to -6 with step 0.5 (thus the

whole grid consisted of 21×25=525 pairs of tested parameters C and γ).
ANN. Neural networks were trained using Mathworks™ Matlab Neural Network Toolbox™ with the same default parameters such as using hyperbolic tangent sigmoid as activation function, applying Levenberg-Marquardt backpropagation training algorithm and selecting the ratio for training:validation:test as 70:15:15 for early stopping on cross-entropy, etc. For each N-dimensional input, the architecture of ANN was selected [with several hidden layers (from one to five) with the number of neurons in it ranging from 5 to 100 (a separate ANN was trained for each of these architecture choices final architecture was then selected based on the best performance). Note that we do not claim that these particular architecture choices for SVM and ANN are optimal; our point is to demonstrate the significant potential of LUPI improvement with different classification methods, whether these methods are optimal or not.

The averaged (over 20 realizations) error rates for these scenarios are shown in Table 6 (for SVM) and in Table 7 (for ANN). The collected results show that performance of SVM (and its LUPI modifications) is better than that of ANN (and its LUPI modifications). They also show that both linear and nonlinear versions of Knowledge Transfer LUPI improve the performance of SVM and ANN on decision inputs (often significantly, in relative terms) in all of the considered scenarios. Note that both versions are just examples of knowledge transfer and other mappings (especially if relevant domain knowledge is available) could be leveraged.

Table 6. Performance of SVM and LUPI on Modified "Parkinsons" Example.

k	SVM on decision features	LUPI (linear)	LUPI (nonlinear)	SVM on all features	LUPI gain (linear)	LUPI gain (nonlinear)
12	13.26%	9.18%	11.32%	6.63%	61.55%	29.25%
11	13.52%	10.66%	12.70%	6.63%	41.49%	11.85%
10	13.16%	10.00%	12.19%	6.63%	48.45%	14.85%
9	12.70%	8.67%	10.76%	6.63%	66.41%	31.95%
8	12.81%	8.52%	10.76%	6.63%	69.44%	33.07%
7	14.49%	11.07%	13.16%	6.63%	43.51%	16.88%
6	13.78%	11.17%	12.35%	6.63%	36.43%	20.01%
5	10.56%	8.98%	9.49%	6.63%	40.27%	27.28%
4	11.22%	10.36%	10.10%	6.63%	18.88%	24.42%
3	12.04%	9.59%	9.44%	6.63%	45.28%	48.13%
2	8.47%	7.55%	7.04%	6.63%	49.99%	77.76%

Table 7. Performance of ANN and LUPI on Modified "Parkinsons" Example.

K	ANN on decision features	LUPI (linear)	LUPI (nonlinear)	ANn on all features	LUPI gain (linear)	LUPI gain (nonlinear)
12	19.49%	16.43%	15.46%	8.01%	26.66%	35.11%
11	19.44%	15.20%	15.56%	8.01%	37.05%	33.93%
10	21.33%	14.64%	15.66%	8.01%	50.18%	42.52%
9	20.66%	12.70%	13.72%	8.01%	62.90%	54.83%
8	20.26%	12.04%	13.98%	8.01%	67.08%	51.25%
7	18.57%	13.01%	15.05%	8.01%	52.65%	33.33%
6	20.20%	13.83%	13.93%	8.01%	52.29%	51.45%
5	16.84%	11.63%	11.27%	8.01%	58.96%	63.00%
4	17.35%	12.45%	12.50%	8.01%	52.46%	51.91%
3	12.14%	11.48%	11.48%	8.01%	16.05%	16.05%
2	10.97%	10.25%	10.25%	8.01%	24.13%	24.15%

Numerically, the error rates of LUPI are between the corresponding SVM or ANN constructed on decision features and on all features. In other words, if the error rate of the algorithm on decision features is B, while the error rate of the algorithm on all features is C, the error rate A of LUPI satisfies the bounds $C<A<B$. So one can evaluate the efficiency of LUPI approach by computing the metric $(B-A)/(B-C)$, which describes how much of the performance gap $B-C$ can be recovered by LUPI. For SVM, this metric varies between 12% and 78%; for ANN, this metric varies between 16% and 67%. Generally, in realistic examples, the typical value for this LUPI efficiency metric is in the ballpark of 35%. Also note that if the gap $B-C$ is small compared to C, it means that the privileged information is not particularly relevant; in that case, it is likely hopeless to apply LUPI anyway: there is little space for improvement for that. It is probably safe to start looking for LUPI solution if the gap $B-C$ is at least 1.5−2 times larger than C.

We have also implemented C-mapping for SVM for the already described datasets using the same setting as for A-mapping, with the following modifications instead of constructing regressions to privileged features, we constructed (positive linear or nonlinear kernel) regressions to functions $K^*(x_i^*, x^*)$ with subsequent selection of top 40 or them, in terms of their relevance to the label, as was determined by RandomForest method.

The averaged (over 20 realizations) error rates for these scenarios are shown in Table 8. The collected results show that both linear and nonlinear versions of Knowledge Transfer LUPI with C-mapping improve the performance of SVM on decision inputs (often significantly, in relative terms) in all of the considered scenarios.

Table 8. Performance of SVM and LUPI on Modified "Parkinsons" Example.

k	SVM on decision features	LUPI (linear)	LUPI (nonlinear)	SVM on all features	LUPI gain (linear)	LUPI gain (nonlinear)
12	13.26%	9.59%	9.59%	6.63%	53.35%	55.35%
11	13.52%	9.79%	10.10%	6.63%	54.14%	49.64%
10	13.16%	11.22%	9.79%	6.63%	29.71%	51.61%
9	12.70%	10.30%	10.51%	6.63%	39.54%	36.08%
8	12.81%	10.20%	10.20%	6.63%	42.23%	42.23%
7	14.49%	9.49%	11.53%	6.63%	63.61%	37.66%
6	13.78%	10.71%	11.94%	6.63%	42.94%	25.73%
5	10.56%	9.18%	10.20%	6.63%	35.11%	9.16%
4	11.22%	8.26%	10.30%	6.63%	64.49%	20.04%
3	12.04%	9.08%	10.41%	6.63%	54.71%	30.13%
2	8.47%	7.57%	8.18%	6.63%	48.91%	15.76%

In this paper, we described several properties of privileged information including its role in machine learning, its structure, and its applications. We extended the existing knowledge transfer research in the area of privileged information (initially considered for SVM) to neural networks and presented a scalable algorithmic framework, which has the same scalability properties as current implementations. The described framework is the first step in the proposed direction, and its further improvements (especially concerning alternative methods of knowledge transfer) will be the subject of future work.

3.3 Software for LUPI

The current distribution software for knowledge transfer LUPI consists of the files

1) priv_predict.py
2) std_predict.py
3) lupi_predict.py
4) SVMstd.py
5) SVMpriv.py
6) SVMlupi.py
7) test_error.py
8) partition.py
9) experiment.py
10) mamiStd.py
11) mamiLupi.py

and the folders

1) data
2) models

In order to run this software, Anaconda has to be installed, which could be done via https://docs.continuum.io/anaconda/install. Once Anaconda is installed, the scripts could be used as follows. First, train and test datasets can be created using the partition.py script:

```
python partition.py
```

which will create train and test datasets for the features X and the labels y. Options of the script can be used to choose the features and label files, the test size and other parameter. If no options are passed to the script, default options are used and both files are saved in ./data/partitions/. Standard SVM is trained by typing the command

```
python SVMstd.py
```

This command will save several files in the ./models/ directory; these files are the models that are needed for prediction. After that, labels for the test file (saved as ./data/partitions/X_test.data) can be obtained by typing the command

```
python std_predict.py
```

which will create a file in ./prediction with the predicted labels, saved as prediction.data. The prediction error then can be computed by comparison to the real labels (which are saved as./data/partitions/y_test.data) by typing the command:

```
python test_error.py
```

The same procedure can be repeated for Privileged SVM as

```
python partition.py --featFile ./data/priv.data
python SVMpriv.py
python priv_predict.py
python test_error.py
```

Finally, knowledge transfer LUPI SVM can be executed as follows (assuming, for this example, that the first 14 features in the dataset are standard, while other features are privileged):

```
python partition.py --featFile ./data/priv.data --lupi 14
python SVMlupi.py
python lupi_predict.py
python test_error.py
```

The options of the described scripts are as follows.

partition.py

- featFile: path to the features file. This file should contain only the features, not the labels. If the file is intended to be used with LUPI, then the standard features should appear first followed by the privileged ones. Default: ./data/std.data
- labelFile: path to the label file. It is assumed that the *n*-th line contains the label of the *n*-th line of features in the featFile. The labels are typically represented by 1 and 0 but in general any two integers can be used instead. Default: ./data/labels.data

- testSize: defines the integer size of the test set. For example, if the number 25 is passed, it will save 25 random data points for the test dataset and put the remaining data points into the training set. Default allocates 25% of the data to the test set.
- seed: random seed to decide the train/test split, it should be an integer. Default is 0.
- outputFile: path to the directory in which the train/test datasets will be saved. Default is ./data/partitions
- lupi: used to indicate that the data are intended for LUPI approach, in which case the train set will contain both standard and privileged features, but the test set will contain only standard ones. If used it expects an integer to indicate how many standard features there are (which are assumed to appear first in the featFile). For example, it the number 5 is passed, it assumes that the data folder has at least 6 columns and the first 5 ones are standard. Default is not to use LUPI approach.

SVMstd.py

- nJobs: number of processors to be used. Expects a positive integer and assumes the machine has that many processors. Default is 1.
- seed: random seed to decide the cross-validation split, it should be an integer. Default is 0.
- kernel: the choice of SVM kernel. It can be either `linear' or `rbf'. When passed as an argument the corresponding word has to be present, e.g. --kernel linear. Default is 'rbf'.
- C_range: defines the range of values for the penalty parameter *C* of the SVM; *C* needs to be a positive number. This option allows the user to pass 3 float numbers: minimum C , maximum C and step. These will be used to construct the following interval: [2^(minC), 2^(maxC)] in steps of 2^(step), not including 2^(maxC). To pass in the values separate them by space, e.g. --*C_range 1 2 0.1* , Default: -5 5.5 .5
- gamma_range: defines the range of values for the gamma parameter of the SVM, when rbf kernel is selected. This is the width of the kernel function. Typically one gives to the machine a range of values that gamma can take and the machine does cross-validation to figure out which value has the best generalization performance. This option allows the user to pass 3 float numbers: minimum gamma , maximum gamma and step. These will be used to construct the following interval: [1/ [2 [2^(min_gamma)]^2, 1/ [2 [2^(max_gamma)]^2] in steps of 1/ [2 [2^(step)]^2], not including 1/ [2 [2^(max_gamma)]^2]. To pass in the values separate them by space, e.g. --*gamma_range 1 2 0.1* , Default: -6 5.5 .5

SVMpriv.py – same as SVMstd.py
SVMlupi.py - same as SVMstd.py plus the following options:

- lupiRegr: the kind of regression for reconstructing the privileged features at test time. This can be any of 'linear, ridge, svr', where linear is standard linear regression, ridge is a kernel ridge non-linear regression, and svr is a support vector regression (also non-linear). Both non-linear regressions need to find optimal parameters during training and hence take a longer time to train. Both use an rbf kernel. When passing an argument, the corresponding word has to be typed, e.g. --lupiRegr linear. Default: ridge
- nStdFeat: number of standard features present, it expects a positive integer. Default: 14

std_predict.py - saves the prediction at ./predictions/prediction.data
priv_predict.py - same as std_predict.py
lupi_predict.py - same as std_predict.py

For the same MAMI set described above, we now describe how to compare LUPI SVM versus Standard SVM. Using the contents of the folder we can run side by side SVM standard and LUPI SVM sufficiently many times to achieve reliable results when we aggregate our findings. For this we provide 50 partitions of the MAMI dataset and two scripts that can automate the experiment. We would like to see the difference between Standard and LUPI as we vary the size of the training sample. Hence we include in our folder 4 subfolders named mami80, mami160, mami320, and mami640. Each contains data partitioned in train and test. Mami80 contains 50 random pairs of train/test sets, where the train sets have size 54 and test sets have size 802. These numbers are determined by the maximum size of data with balanced class labels if we start with 80 (or 160/320/640) random samples. The remaining data points are put in the test set. Similarly mami160 has a 108/748 split, mami320 has a 216/640 split, and mami640 has a 432/424 split.

These ready partitioned data allow us to run experiments and compare Standard and LUPI SVM for different training sizes of 54, 108, 216 and 432 data points.

To run these experiments, we used the mamiStd.py script as follows, to see the performance of the SVM Standard algorithm:

```
python mamiStd.py --nJobs 2 --trainSize 80
```

where nJobs tell the computer to use 2 processors and trainSize tells it to run the experiment for size 52. Similarly we can run the experiments for the rest of the train size. The output of the algorithm is the average test error across the 50 experiments.

To run the corresponding experiments for LUPI use mamiLupi.py script as follows:

```
python mamiLupi.py --nJobs 2 --trainSize 80
```

where nJobs tell the computer to use 2 processors and trainSize tells it to run the experiment for size 52. Similarly we can run the experiments for the rest of the train size. The output of the algorithm is the average test error across the 50 experiments.

We have provided a special script that can automate the different steps of a full train/test cycle. The experiment.py script can be used to run a number of experiments on any dataset and get the average test error. Such repetitive experiments are necessary in order to get better approximation of the true test error for a learner. Below we explain the option features for the experiment script.

experiment.py

- featFile: path to the features file. This should contain only the features, not the labels. If it's intended to be used with LUPI, then the standard features should appear first followed by the privileged. Default: ./data/std.data
- labelFile: path to label file. It's assumed that the n-th line contains the label of the n-th line of features in the featFile. The labels are typically represented by 1 and 0 but in general any two integers can be used instead. Default: ./data/labels.data
- testSize: defines the size of the test set, as an integer. If the number 25 is passed, it will save 25 random data points for test and put the rest of the data points in the train set. Default: gives 25% of the data to the test set

- lupi: used to indicate that the data are intended for a LUPI procedure, in which case the train set will contain standard and privileged features, but the test will contain only standard. If used it expects an integer to indicate how many standard features there are (which are assumed to appear first in the featFile). If the number 5 is passed, it is assumed that the data folder has at least 6 columns and the first 5 are standard. Default: assumes lupi is not being done.
- nJobs: number of processors to be used. Expects a positive integer and assume the machine has that many processors. Default: 1
- kernel: the choice of SVM kernel. It can be either `linear' or `rbf'. When passed as an argument, the corresponding word has to be type, e.g. --kernel linear. Default: rbf
- C_range: defines the range of values for the C parameter of the SVM. This is the tolerance of error when constructing the classification boundary. C needs to be a positive number, but typically one gives to the machine a range of values that C can take and the machine does cross-validation to figure out which value has the best generalization performance. This option allows the user to pass 3 float numbers: minimum C , maximum C and step. These will be used to construct the following interval: [2^(minC), 2^(maxC)] in steps of 2^(step), not including 2^(maxC). To pass in the values separate them by space, e.g. --*C_range 1 2 0.1* , Default: -5 5.5 .5
- gamma_range: defines the range of values for the gamma parameter of the SVM, when rbf kernel is selected. This is the width of the kernel function. Typically one gives to the machine a range of values that gamma can take and the machine does cross-validation to figure out which value has the best generalization performance. This option allows the user to pass 3 float numbers: minimum gamma , maximum gamma and step. These will be used to construct the following interval: [1/ [2 [2^(min_gamma)]^2, 1/ [2 [2^(max_gamma)]^2] in steps of 1/ [2 [2^(step)]^2], not including 1/ [2 [2^(max_gamma)]^2]. To pass in the values separate them by space, e.g. --*gamma_range 1 2 0.1* , Default: -6 5.5 .5
- lupiRegr: (used only together with lupi option) the kind of regression for reconstructing the privileged features at test time. This can be any of 'linear, ridge, svr', where linear is standard linear regression, ridge is a kernel ridge non-linear regression, and svr is a support vector regression (also non-linear). Both non-linear regressions need to find optimal parameters during training and hence take a longer time to train. Both use an rbf kernel. When passed as an argument, the corresponding word has to be typed, e.g. --lupiRegr linear. Default: ridge

Here we explain how to use this software on a different dataset. The most important thing is to save the data in the appropriate format. Then direct the scripts have to be directed to the saved datasets. The format of the data has to be following:

Features file - a csv file that contains columns of data, where each column is a feature. If some of these features are designated as Standard features, for LUPI purposes, then these Standard features should appear first, followed by the rest of the features, which will be assumed to be privileged. For example we have provided two different such files in the data folder and. We have saved as priv.data all the mami dataset with the first 14 features being the standard features, in ./data/priv.data. We have also saved just the standard features as a separate folder in ./data/std.data

Labels file: a csv that contains line separated integers that represent the class labels of each examples from the features file (in the same order).

Once the two files are saved, say features.data and labels.data he partition.py script has to be pointed to these two files to be split into train and test set as follows:

python partition.py --featFile <path to features.data> --labelFile <path to labels.data>

More options can be included, such as the test size or whether the data are supposed to be used with LUPI (in which case, the option --lupi <number of standard features> should be used). Once train and test partitions are saved, SVMs can be trained on these data using the same instructions as before.

4 RESULTS AND DISCUSSIONS

As described in the previous section, we proposed and developed a novel framework for most accurate computation of key statistical elements of model-driven problems (such as conditional probability, regression, etc.). The proposed rigorous approach avoids the constraints of traditional model-driven approaches; instead, the problem of computing the corresponding statistical quantities is formulated according to its definition, which involves the corresponding Fredholm integral equation. This ill-posed equation, upon replacing unknown quantities with their data-driven empirical approximations, are converted to the corresponding quadratic programming problems, which could be solved with standard solvers. Additional domain knowledge components could be encoded as constraints for these quadratic programming problems.

This framework was developed in the course of PPAML program and described in detail in our publications [1] [2] [3]. The results, described in these papers, demonstrate the viability of the proposed approach and superior performance as compared to classical methods.

For more narrow, but very important class of problems, where the assumption of the monotonicity of the results is essential (a typical classifier, such as SVM or ANN, satisfies this assumption), additional Synergy methodology was developed. This methodology is based on a rigorous approach of merging results of different decisions (classifiers) in the most accurate way. It was developed in the course of PPAML program and described in detail in [13]. It is applicable to ensemble methods (merging outputs of different classifiers), bagging (merging outputs of classifiers trained on different subsets of the original training set), parameter selection (merging outputs of classifiers trained on different areas of the parameter space), etc. Performance improvements, achieved by the propose mechanisms, have been varying up to 35% improvement of accuracy over SoA (standard ensemble methods).

As described in the previous section, we developed a solid and scalable mechanism for capturing domain knowledge in the form of features and kernels for standard data-driven problems. This mechanism was implemented for learning using privileged information. The implemented techniques for encoding model-based information into features with improved performance by 40% over SoA (standard SVM and neural networks). We have also developed LUPI implementation in Python and released it as open source code.

More details about the outlined methods and approaches can be found in the following nine publications that were published based on results of our research in DARPA PPAML program:

(1) V.Vapnik, I.Braga, R.Izmailov, Constructive Setting for Problems of Density Ratio Estimation, Statistical Analysis and Data Mining, vol. 8, no. 3, June 2015, pp. 137-146.

(2) V.Vapnik, R.Izmailov, Statistical Inference Problems and Their Rigorous Solutions, in Statistical Learning and Data Sciences, A.Gammerman, V.Vovk, H.Papadopulos (Eds). Lecture Notes in Artificial Intelligence 9047. Proceedings of Third International Symposium, SLDS. London, Springer, 2015, pp.33-71.

(3) V.Vapnik, R.Izmailov, V-Matrix Method of Solving Statistical Inference Problems, Journal of Machine Learning Research, 16:1683-1730, 2015.
(4) V.Vapnik, R.Izmailov, Synergy of Monotonic Rules, Journal of Machine Learning Research, 17:1-33, 2016.
(5) V.Vapnik, R.Izmailov, Learning Using Privileged Information: Similarity Control and Knowledge Transfer, Journal of Machine Learning Research, 16:2023-2049, 2015.
(6) V.Vapnik, R.Izmailov, Learning with Intelligent Teacher: Similarity Control and Knowledge Transfer, in Statistical Learning and Data Sciences, A.Gammerman, V.Vovk, H.Papadopulos (Eds). Lecture Notes in Artificial Intelligence 9047. Proceedings of Third International Symposium, SLDS. London, Springer, 2015, pp.3-32.
(7) V.Vapnik, R.Izmailov, Learning with Intelligent Teacher, in Lecture Notes in Artificial Intelligence 9653. Proceedings of 5th International Symposium, COPA 2016. Springer, 2016.
(8) R.Ilin, R.Izmailov, Y.Goncharov, S.Streltsov, Fusion of Privileged Features for Efficient Classifier Training, 19th International Conference on Information Fusion, pp.1-8, 2016.
(9) V.Vapnik, R.Izmailov, "Knowledge Transfer in SVM and Neural Networks", Annals of Mathematics and Artificial Intelligence, 1-17, 2017.

5 CONCLUSIONS

We have successfully developed two complementary techniques to standard machine learning approaches (model-driven and data-driven) by concentrating on their drawbacks and addressing them with advantages of the complementary approach. Specifically, we developed novel data-driven techniques for improving model-driven approach, and, conversely, novel model-driven techniques for improving data-driven approach. Both developments are implemented as scalable algorithms, and published in nine papers in academic journals and conferences.

6 REFERENCES

1. V.Vapnik, I.Braga, R.Izmailov, Constructive Setting for Problems of Density Ratio Estimation, Statistical Analysis and Data Mining, vol. 8, no. 3, June 2015, pp. 137-146.
2. V.Vapnik, R.Izmailov, Statistical Inference Problems and Their Rigorous Solutions, in Statistical Learning and Data Sciences, A.Gammerman, V.Vovk, H.Papadopulos (Eds). Lecture Notes in Artificial Intelligence 9047. Proceedings of Third International Symposium, SLDS. London, Springer, 2015, pp.33-71.
3. V.Vapnik, R.Izmailov, V-Matrix Method of Solving Statistical Inference Problems, Journal of Machine Learning Research, 16:1683-1730, 2015.
4. J. Platt (1999). Fast training of Support Vector Machines using Sequential Minimal Optimization. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pp. 185–208.
5. H. Lin, C. Lin, and R. Weng. A note on Platt's probabilistic outputs for support vector machines. *Machine Learning*, 68 (3): 267–276, 2007.

6. T. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems, LBCS-1857*, pages 1–15. Springer, 2000.
7. C. Zhang and Y. Ma. *Ensemble Machine Learning: Methods and Applications*. Springer New York, 2012.
8. A. Tsybakov. *Optimal Rates of Aggregation*, pages 303–313. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
9. V. Vapnik (1998). *Statistical Learning Theory*, Wiley-InterScience.
10. R. Izmailov, V. Vapnik, and A. Vashist (2013). Multidimensional splines with Infinite Number of Knots as SVM Kernels. Proc. *International Joint Conf. on Neural Networks (IJCNN)*.
11. S. Wang, A. Mathew, Y. Chen, L. Xi, L. Ma, and J. Lee. Empirical analysis of support vector machine ensemble classifiers. *Expert Systems with Applications*, 36 (3 Pt2): 6466–6476, 2009.
12. M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.
13. V.Vapnik, R.Izmailov, Synergy of Monotonic Rules, Journal of Machine Learning Research, 17:1-33, 2016.
14. V.Vapnik, R.Izmailov, Learning Using Privileged Information: Similarity Control and Knowledge Transfer, Journal of Machine Learning Research, 16:2023-2049, 2015.
15. V.Vapnik, R.Izmailov, Learning with Intelligent Teacher: Similarity Control and Knowledge Transfer, in Statistical Learning and Data Sciences, A.Gammerman, V.Vovk, H.Papadopulos (Eds). Lecture Notes in Artificial Intelligence 9047. Proceedings of Third International Symposium, SLDS. London, Springer, 2015, pp.3-32.
16. V.Vapnik, R.Izmailov, Learning with Intelligent Teacher, in Lecture Notes in Artificial Intelligence 9653. Proceedings of 5th International Symposium, COPA 2016. Springer, 2016.
17. R.Ilin, R.Izmailov, Y.Goncharov, S.Streltsov, Fusion of Privileged Features for Efficient Classifier Training, 19th International Conference on Information Fusion, pp.1-8, 2016.
18. V.Vapnik, R.Izmailov, "Knowledge Transfer in SVM and Neural Networks", Annals of Mathematics and Artificial Intelligence, 1-17, 2017.
19. V. Vapnik (1998). *Statistical Learning Theory*, Wiley-InterScience.
20. V. Vapnik (2006). *Estimation of Dependences Based on Empirical Data*. 1st Edition 1982; 2nd Edition 2006. Springer.
21. V. Vapnik and A. Vashist (2009). A New Learning Paradigm: Learning Using Privileged Information. *Neural Networks*, vol. 22, pp. 544-555.
22. D. Pechyony, R. Izmailov, A. Vashist, and V. Vapnik (2010). SMO-Style Algorithm for Learning Using Privileged Information. The 6th *International Conference on Data Mining (DMIN)*, pp. 231-245.
23. S.Fouad, P.Tino, S.Raychaudhury, and P.Schneider, "Incorporating privileged information through metric learning," IEEE Transactions on Neural Networks and Learning Systems 24 (2013), pp.1086-1098.
24. R.Ilin, S.Streltsov, and R.Izmailov, "Learning with privileged information for improved target classification," International Journal of Monitoring and Surveillance Technologies Research, 2(3), 5-66, July 2014.
25. B.Ribeiro, C.Silva, N.Chen, and A.Vieirac, J.Carvalho das Nevesd, "Enhanced default risk models with SVM+," Expert Systems with Applications 39 (2012), pp.10140–10152. V. Sharmanska and C.Lampert, "Learning to rank using privileged information," International Conference on Computer Vision (ICCV). IEEE, 2013, pp.825-832.
26. P. Bendlich, E.Gasparovic, J.Harer, R.Izmailov, L. Ness (2015), Multi-Scale Local Shape Analysis and Feature Selection in Machine Learning Applications, Proceedings of International Joint Conference on Neural Networks (IJCNN), 2015.

27. R. Ilin, S. Clouse, Extraction and Classification of Moving Targets in multi-sensory MAMI-1 Data Collection, NAECON 2015.
28. A. Freeman, A. L. Cain, H. Zelnio, E.Watson, and O. Mendoza-Schrock, “Minor area motion imagery (MAMI) dismount tower data challenge problems,” IEEE National Aerospace and Electronics Conference, NAECON 2014 , pp. 224 – 227.
29. E.Blasch, H.Ling, Y.Wu, G.Seetharaman, M.Talbert, L.Bai, and G.Chen, “Dismount tracking and identification from electro-optical imagery,” SPIE Conference on Defense Security+Sensing, 2012.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

SVM Support Vector Machines
ANN Artificial Neural Network